Universities Press

COMPUTER SCIENCE

DATA MINING

TECHNIQUES

33

Arun K Pujari

DATA MINING

TECHNIQUES

Arun K Pujari

Universities Press

Universities Press (India) Private Limited

Registered Office
3-5-747/1/A & 3-6-754/1 Himayatnagar
Hyderabad 500 029 (A.P.), India
e-mail: info@universitiespress.com

First published 2001
Tenth Impression 2008

ISBN-13: 978 81 7371 380 4
ISBN-10: 81 7371 380 4

Typeset by
OSDATA
Hyderabad 500 029

Printed in India at
Graphica Printers
Hyderabad 500 013

Published by
Universities Press (India) Private Limited
3-5-747/1/A & 3-6-754/1 Himayatnagar
Hyderabad 500 029 (A.P.), India

To
my mother

Late Smt. Tilottama Devi

— *A very fragile and weak lady who was the pillar of strength for all her children and her husband to lean on*

Contents at a Glance

List of Contents

FOREWORD

Vast amounts of operational data are routinely collected and stored away in the archives of many organizations. To take a simple example, the railway reservation system has been operational for over a decade and a vast amount of data is generated each day on train bookings. Much of this data is probably archived for audit purposes. This archived operational data can be effectively used for tactical strategic management of the railways. For instance, by analyzing the reservation data it would be possible to find out traffic patterns in various sectors and use it to add or remove bogies in certain trains, to decide on the mix of various classes of accommodation, etc. This, however, was not feasible until recently due to limitations in both hardware and software. In the last five years there has been a tremendous improvement in hardware—512 MB of main memory, 40 GB disk and CPU clock of 1.6 GHZ is now available on a PC. Thus, computer programs which sift massive amounts of operational data, recognize patterns and provide hints to formulate hypotheses for tactical and strategic decision-making can now be executed in a reasonable time. This has opened up a fertile area of research to formulate appropriate algorithms for mining archival data to formulate and test hypotheses. An array of ideas from computer science, statistics and management science are being applied in this area. This subject is currently an active research area.

I am extremely happy that Professor Arun K. Pujari has written this book on Data Mining Techniques. It is a timely addition to the existing computer science literature and will be a boon to students. Professor Pujari has been an active researcher in this area. He has also introduced a course on Data Mining at the University of Hyderabad and has been teaching it. His rich experience in teaching and research is evident in the selection of topics and their treatment in this book. The book starts with a brief introduction followed by a detailed chapter on data warehousing. It has chapters which discuss a large number of diverse algorithms using association rules, clustering techniques, decision tree techniques, neutral networks, genetics, rough sets, and support vector machines. A novel aspect of this book is its discussion of the emerging area of web mining of textual data. A large number of references are cited and the treatment is lucid and up to date. I recommend this book to students who want to explore this important subject.

V. Rajaraman
Honarary Professor
Super Computer Education & Research Center
Indian Institue of Science, Bangalore

PROLOGUE

In this Information Age, the data generated by the day-to-day operations constitute one of the powerful assets of an enterprise. Most enterprises generate unlimited amounts of data from their On-Line Transaction Processing (OLTP) systems, Point-of-Service (POS) systems, financial ATMs and the Web. The challenge faced by these enterprises with regard to the massive data-rich but information-poor collection is to extract valuable *information* to be available at a particular time, place and in the form needed to support the decision-making process.

Scientific organizations also generate large volumes of data. Artificial satellites are streaming in volumes of remote sensing data every minute. Worldwide activities on the genome and related projects are resulting in massive technical material. These are only two examples of modern technology resulting in valuable data of a highly specialized nature. The challenge is to analyze the data to arrive at meaningful conclusions.

Techniques like clustering, decision tree, and NN are known in pattern recognition, statistics and other disciplines and several algorithms have been devised. However, these algorithms are not suitable for the purpose of mining and hence new algorithms had to be proposed in recent years for this purpose.

This book deals with those algorithms that are specially designed for data mining. It begins with an introduction to the basic concepts of data mining and gradually brings the reader to the frontiers of research on this topic. It discusses all the major and current techniques of data mining. Specific mining problems such as web mining, text mining, temporal mining and spatial mining will be of great interest to researchers who are already familiar with the basic mining techniques.

Professor Pujari, a distinguished Professor of Computer Science in our University, with over 10 years of teaching and research experience in our University, has striven hard to put together this valuable monograph. I am sure that besides students in Computer Science, those in Mathematical Science and Management Studies will find it immensely useful. I wish this publication every success.

P. Rama Rao
Vice-Chancellor
University of Hyderabad

PREFACE

Data mining and Data Warehousing are increasingly becoming popular among IT professionals, academics and pupils. In the last few years, almost in every meeting which has anything to do with Databases, Neural Networks, Genetic Algorithms, E-Commerce, or Artificial Intelligence has had a theme or session on Data Mining and Warehousing. Researchers from different disciplines are gradually being attracted to work in this new frontier of research. Business enterprises-small-, medium- or large-scale ones are considering the deployment of a warehouse as a major progressive step and as a matter of pride. Business Intelligence, Customer Relation Management(CRM) and E-commerce all hinge on Data Mining and Warehousing. Universities and other training institutions, keeping pace with technological development, are also beginning to introduce this subject in their curricula. These two terms are now justly becoming household terms.

During 1997–98, many of the universities in US and Europe were introducing this as a course at the graduate level. To keep abreast with this development, I started an optional course on data mining and warehousing during the winter semester of 1997–98 at the University of Hyderabad. Since then, this course has become increasingly popular. I had to depend on the collection of research papers as the course material and my own compilation of these results became the reading material for the course. Recently, many other universities decided to introduce this as an optional course for the degree as well as postgraduate level. This development and the self-compiled lecture notes prompted me to venture to prepare this book.

My aim is to present all the major techniques of Data Mining and Warehousing in a comprehensible manner, so that it can serve not only as a textbook but also a handbook for any researcher. Efforts have been made to include the most recent algorithms (till the latter half of 2000). I have tried to include all the major techniques of data mining here. Some of the recent techniques like Support Vector Machines, Rough Set Theory, etc. are also introduced as tools of data mining. I feel that it will be useful for many researchers and students in Computer Science, Management Science and Mathematical Sciences. The present text is envisaged as a textbook as well as a research handbook in the areas of data mining and data warehousing. Since I am trying to address heterogenous categories of readers, there is some unevenness in the presentation styles of the chapters. Chapters 2, 3, 4, 5 and 6 are presented in a textbook style for the benefit of any graduate-level student. In Chapter 7, my intention is to

apprise the reader about the basic (introductory level) concepts of NN, GA, Rough Sets and SVM and to show that these techniques are also important ones for Data Mining. The content of this chapter is, though self-explanatory, not very elaborate like other chapters. Readers who are interested in learning the algorithmic details of these techniques may have to refer to other books/articles addressing the specific technique. Chapter 8 and Chapter 9 are meant for advanced-level readers. In future, I intend prepare materials elaborating the details of the contents of Chapters 7, 8 and 9.

Readers are requested to send their comments and suggestions on the text without hesitation.

Arun K Pujari
akpcs@uohyd.ernet.in
February 2001

ACKNOWLEDGEMENT

Writing a textbook is not an easy task. If this book has taken shape, it is primarily because of persons around me who motivated, encouraged and helped me to ensure that this project reached the stage of completion. I am indebted to a number of individuals in academic as well as in social circles who have contributed in different, but important, ways in the preparation of this book. I take this opportunity to express my gratitude to all those who have helped and encouraged me to prepare this book.

I am also indebted to Professor Gautam Das who introduced me to some of the newer areas of data mining. My thanks go to Professor Malaya Dutta, Dr Dhruba Bhattacharya, and Dr Rajiv K Das of Tezpur University, for going through the first draft of the book and pointing out the errors.

This book grew out of lecture notes for the course on Data Mining and Data Warehousing introduced by me in the Department of Computer & Information Sciences, University of Hyderabad. Thanks are also due to the three batches (1997–98, 1998–99 and 1999–2000) of 100+ students who have opted for this course and patiently sat through it. They have provided their most helpful assistance in the improvement of the text material. Special thanks are due to Suresh Reddy, Rajesh, Priyambada, Atul, Karthik, Sivaramakrishna, and many other students who have pointed out errors in the text. In addition, I am indebted to a large number of masters and doctoral students who helped me in implementing new ideas and applications in this area. This contributed significantly towards my understanding of the subject. I am thankful to Adilakshmi, Shailaja, Vijaya Kumari, and Dhananjay Naidu for helping me in proofreading, formatting, diagrams, etc., during the preparation of this manuscript. While preparing the text, I gave talks at seminars on some of the chapters at different institutions. I am grateful to IIM/Calcutta, IDRBT/Hyderabad, ADRIN/Hyderabad, and Tezpur University for this.

I acknowledge gratefully the authors of innumerable research papers who have promptly mailed me copies of their research articles at my request and also to the authors for making their papers downloadable. I owe my gratitude to University of Hyderabad for granting me sabbatical leave to prepare the book. I am grateful to Professor P Rama Rao, the Vice-Chancellor who has been constantly encouraging and agreeing to write the prologue for this book. Many organizations have extended their facilities to me during the preparation of the book. These are the University of Hyderabad, the University of Memphis, and Griffith University.

Writing a book involves a major commitment of time and energy and a commensurate loss of availability of that time and energy for any other purpose. All my doctoral students – Sreenivasa Rao, Vijaya Kumari, Durga Bhavani, Adilaxmi and Jayasree – had to bear with me for not being able to attend to their dissertation work. The love, patience, understanding, and encouragement of my wife Sheila and our son Pinak are greatly appreciated. Sheila had to play many roles during this period—taking dictation, formatting, and editing the manuscript. I often found myself in the difficult situation of having to answer Pinak's query regarding the progress of the book during his weekend visits from the hostel. The early influences of my parents and my teachers are also reflected in the presentation style.

Finally, I am grateful to Universities Press for agreeing to publish this book. Mr Madhu Reddy's personal interest in the book has helped me to complete the manuscript on time. My special thanks to Ms Chandini Rao for patiently going through the material repeatedly for corrections. I am also very grateful to Professor V Rajaraman for agreeing to comment on the manuscript and for writing the Foreword.

Arun K Pujari
Hyderabad
February 2001

1

INTRODUCTION

1.1 INTRODUCTION

The advent of computing technology has significantly influenced our lives and two major impacts of this effect are Business Data Processing and Scientific Computing. During the initial years of the development of computer techniques for business, computer professionals were concerned with designing files to store the data so that information could be efficiently retrieved. There were restrictions on storage size for storing data and on the speed of accessing the data. Needless to say, the activity was restricted to a very few, highly qualified professionals. Then came an era when the task was simplified by a DBMS. The responsibility of intricate tasks, such as declarative aspects of the programs were passed on to the database administrator and the user could pose his query in simpler languages such as query languages. Thus, almost any business – small, medium or large scale – began using computers for day-to-day activities. The scenario on the hardware side was encouraging too. The storage cost and machine cost were drastically reduced so that almost any business house could afford a standard machine. Thus, due to widespread computerization and also due to affordable storage facilities, there is an enormous wealth of information embedded in huge databases belonging to different enterprises. Every organization is now accumulating a large volume of daily transaction data and storing then as archival files. As a result, masses and masses of data – megabytes, gigabytes, terabytes – are piling up in the electronic vaults of companies, governments and research institutions.

Business is inherently competitive and in this competitive world of business one is constantly on the lookout for ways to beat the competition. A question that naturally

arose is whether the enormous data that is generated and stored as archives can be used for improving the efficiency of business performance.

In the domain of scientific computing, the major problem is to infer some valuable information from the observed data. The use of computing technology has helped researchers to collect very large volumes of data. The development of other scientific disciplines helped the community to collect such large volumes of data effortlessly. Some typical examples are remote-sensing data, as satellites are streaming in an enormous amount of remote sensing data every day. In the area of health science, the repository of protein data and genome data are an invaluable source of scientific experiments. On the other side, we are drowning in oceans of text data and data of other media that are generated from the Web.

What use are all these data? Up to the early 1990s, the answer to this was 'not much'. No one was really interested in utilizing data which was accumulated during the process of daily activities. Once the transaction processing is over, these data were dumped into archival files.

Such collections of data, whether their origin is business enterprise or scientific experiment, have recently spurred a tremendous interest in the areas of knowledge discovery and data mining. Statisticians and data miners now have faster analysis tools that can help sift and analyze the stockpiles of data, turning up valuable and often surprising information. As a result, a new discipline in Computer Science, Data Mining, gradually evolved. Data mining is the exploration and analysis of large data sets, in order to discover meaningful patterns and rules. The key idea is to find effective ways to combine the computer's power to process data with the human eye's ability to detect patterns. The techniques of data mining are designed for, and work best with, large data sets.

The evolution of data mining began when business data was first stored in computers and technologies were generated to allow users to navigate through the data in real time. Data mining takes this evolutionary process beyond retrospective data access and navigation, to prospective and proactive information delivery. This evolution is due to the support of three technologies that are sufficiently mature: massive data collection, high performance computing and data mining algorithms. The core components of the data mining technologies have been under development for decades, and today the maturity of these techniques coupled with the high performance relational database engines and broad data integration efforts have made these techniques practically applicable.

Data mining is a component of a wider process called *knowledge discovery from databases*. It involves scientists from a wide range of disciplines, including mathematicians, computer scientists and statisticians, as well as those working in fields such as machine learning, artificial intelligence, information retrieval and pattern recognition. Before a data set can be mined, it first has to be cleaned. This removes errors, ensures consistency and takes missing values into account. Data mining may use

quite simple statistical techniques or it may use highly sophisticated data analysis. What is new for data miners is the employment of these techniques on vast quantities of data.

Business enterprises are beginning to realize that information on customers and their buying patterns are the most valuable information. Competitiveness increasingly depends on the quality of decision making and learning to make quality decision from past transactions and decisions. The large databases maintained (generated during the routine operations) by retailers, telecom service providers, and credit card companies contain very valuable information related to customers. These enterprises could benefit immensely in the areas of marketing, advertising and sales, if interesting and previously unknown customer buying and calling patterns can be discovered from the large volumes of data. Several factors have contributed to bring data mining to the forefront. To identify a few of these factors:

- The untapped value in large databases.
- The concept of Data Warehousing
- The dramatic drop in the cost/performance ratio of hardware systems.

The most important of the these three is the development of Data Warehousing technology. Let us take a few minutes to review the layered architecture of the modern database management systems and the modern applications of these DBMSs. The different layers can be viewed as follows:

1. **External Interface** This layer deals with the concepts of data definition language, data manipulation language and host-language interfaces.
2. **Language Processing** Query processing, algebraic optimization, and plan creation are the components of this layer.
3. **Code Generation** The procedural part is handled in this layer in the form of index structures, and the implementation of algebraic operations.
4. **Transaction Management** The important aspects of modern DBMS are concurrency control, logging and recovery.
5. **Storage Management** This layer is concerned with memory and buffer management, secondary storage and device management.

A major change in the above approach is Data Integration. Data integration of data from multiple sources is the key term in modern database applications. Modern applications of database technology demand an integrated access to the multiple databases. Typical applications where data integration is relevant, arise from the merging of different companies, accessing health care data from different institutions, and data access through web-browsing. Recent developments of e-commerce also demand data integration. Data integration can be carried out in two modes, virtual or materialized. The virtual modes of data integration adopt a search engine to collect data from multiple sources, or a mediated

mode to decompose the queries to each of the sources and present the result in a uniform manner. The materialized mode of data integration can be either through a universal DBMS or a data warehouse. In an universal DBMS, data are integrated by physically migrating data from different sources to a new database in OO-DBMS or OR-DBMS. The data warehouse collects data from different sources for OLAP and data mining applications. Among all these modes of integrations, data warehousing is considered to be the most efficient for decision support applications. This is particularly relevant in the contexts of Data Mining and OnLine Analytical Processing (OLAP).

As the terms "Data Mining" and "Data Warehousing' are becoming popular, the last few years have witnessed the emergence of extremely innovative and elegant techniques for data mining and warehousing. Most of these results are presented at different international fora. However, more and more researchers from other areas are interested in knowing about the subject. The present work aims at providing at one place the most recent and important techniques of data mining and warehousing.

1.2 Data Mining as a Subject

Data mining will be of interest to three major streams:

- Statistics
- Computer Science
- Business Management

The course structure and the major thrust of subtopics may differ, based on which of the above groups is the target audience. Thus, there can be three different areas of activities:

- Theory
- Algorithm and implementations
- Applications

The first approach broadly deals with the underlying theoretical concepts of data mining techniques. One attempts to develop a theoretical foundation to extract information from data. The theoretical background of the decision tree, clustering, fuzzy sets, and other statistical techniques, fall into this category of study. A student from mathematics or from statistics may find this approach very interesting.

Just developing the theoretical foundation for data mining and data warehousing will not suffice. One has to also develop the appropriate algorithms to handle large databases. For example, the theory of the decision tree has been known to statistians for quite some time, and it is widely used by the Machine Learning community. But in order to make the decision tree construction process suitable for data mining, it is

necessary to design new algorithms that can handle large databases which are stored in secondary memory. Moreover, one needs to work out the details of integrating the algorithm with a repository of data, either through a DBMS or through a data warehouse. These aspects are more relevant to computer scientists.

The third one is to identify new areas of data mining application for effective business processes. Much of the data mining community comes from an academic background and have focused on algorithms buried deep in the bowels of technology. But algorithms are not what business users care about. Over the past few years the technology of data mining has moved from the research lab to Fortune 500 companies, requiring a significant change in focus. Companies spend millions of dollars to build data warehouses to store their data and data mining applications must take advantage of this. Solving a business problem is much more valuable to a user than solving a statistical modelling problem. This means that a cross-selling specific application is more valuable than a general modelling tool that can create cross-selling models. Different case studies are to be analyzed in detail in this approach to training. A management science student will find such an approach useful.

Thus, we attempt to establish here that the techniques of data mining and data warehousing have grown to such an extent that it can be considered as a full-fledged subject and it can be a part of the training in Statistics, Computer Science and Business Management.

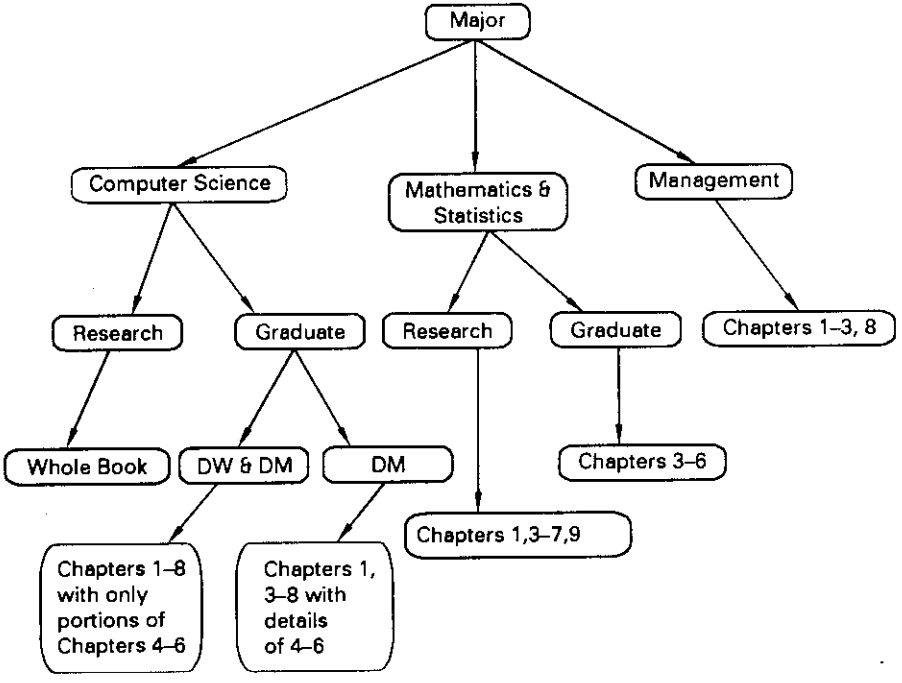
This book is envisaged to cater to needs of these three communities. Of course, Computer Science gets more emphasis in the text. In the following section, we give navigation chart for each of these disciplines.

1.3 Guide to this Book

This section presents a guide for reading this book. The layout of the remainder of the book and the contents of the individual chapters are reviewed briefly. This book is divided into 9 chapters. Chapter 2 presents the basic concepts, architecture and different components of Data Warehousing techniques. It also demonstrates the relevance of Data Mining in the context of Data Warehousing. Chapter 3 introduces the fundamentals of Data Mining. Different definitions and different techniques of Data Mining are discussed in this chapter. Chapter 4 deals with the techniques of discovering Association Rules. It discusses in details all the major algorithms of finding frequent sets for a large database. Chapter 5 provides a detailed survey of clustering techniques for Data Mining. It may be noted that clustering also happens to be a topic in Pattern Recognition and in Statistics. But this book concentrates solely on this technique with regard Data Mining applications. Readers interested in the general theory of clustering technique may require other books for references. Chapter 6 deals with the Decision Tree techniques of Data Mining. It begins with the basic concepts of

decision trees and proceeds to survey the techniques to build decision trees for Data Mining applications. The goal of this chapter is to provide a synthesized exposition of decision trees. Chapter 7 discusses some of the recent trends in Data Mining techniques. Topics like Neural Networks, Genetic Algorithms Rough Set Theory or Support Vector Machines provide very sophisticated tools for Data Mining. But unfortunately researchers in these areas, unlike in the areas of decision trees and clustering, have not yet come up with specific modifications of these theories suitable for large databases. Nevertheless, it is important for a student of Data Mining to study these topics. Chapter 7 introduces these four topics and provides pointers to their possible uses in Data Mining. Mining of data in the Web is as important as mining data in a DBMS or in a warehouse. The Web is a very important source of massive amounts of data. Chapter 8 discusses the techniques of Web mining. Finally, Chapter 9 deals with mining of Temporal and Spatial data.

We give below the decision tree-like structure for the reader to select the appropriate chapters of their interests. For instance, if a reader is a graduate student with CS major and is doing a course on data warehousing and mining then, he could read Chapters 1 through 8, ignoring the details of Chapters 4, 5 and 6.



Decision Tree-like Structure

2

DATA WAREHOUSING

2.1 INTRODUCTION

The advent of Information Technology has affected all walks of life. In the context of business processes, this field has evolved from the good old term *EDP* to *MIS*, and then to *DSS*. When we talk of the use of information technology in a business environment, we cannot ignore the presence of a database system at its core. Database technology, meanwhile, has also grown from a simple file system to a data navigation system, then to a query-based retrieval system, and to an interactive information extraction from distributed sources to support decision-making tasks. Over the last few decades, a majority of top- and middle-level business institutions have adopted the computerization process and, as a result, have been using very large operational databases for their day-to-day activities. Most often, in a large enterprise there exist several operational databases catering to the needs of different

departments. The top managers at the corporate level may like to extract enterprise-wide summarized information to support their decision-making activities. Normally, the operational databases record transaction-like information, and do not explicitly record the summarized information as required by the top-level decision-makers at different points of time. Moreover, corporate managers may like to work in an environment that need not access (or intervene into) the operational database. There is another aspect to such enterprise-wide computing. Different departments of the enterprise design their operational databases taking into their own local requirements, and the availability of their resources. They also typically collect diverse kinds of data and maintain large databases from multiple, heterogeneous, autonomous, and distributed information sources. The integration of such data so as to provide easy and efficient access is highly desirable, and yet challenging. The state-of-art database technology provides us with two different types of tools to extract information from a distributed, heterogeneous environment.

The *traditional database approach* to heterogeneous database integration is to build *wrappers* and *integrators* (or *mediators*) on top of multiple heterogeneous databases. These tools facilitate the following tasks. As and when a query (a requirement of the top-level decision-maker) is posed, a metadata dictionary is used to translate the query into queries appropriate for the individual heterogeneous sites involved. These queries are then mapped and sent to local query processors. The results returned from the different sites are integrated into a global answer set. We may also call this a *query-driven* approach (or some term it a *lazy* approach, or an *on-demand* approach). This query-driven approach requires complex information filtering and integration processes, and competes for resources with processing at local sources. It is inefficient and potentially expensive for frequent queries, especially for queries requiring aggregation.

On the other hand, there exists an interesting alternative to this traditional approach. We term it the *update-driven* approach, *eager* approach or, *in-advance* approach. In an update-driven approach information from multiple, heterogeneous sources is integrated in advance and stored separately for direct querying and analysis. Though it does not contain the most current information, it brings high performance. Furthermore, query processing in this model does not interfere with the processing at local sources.

Data warehousing is essentially the process that facilitates the update-driven approach. It is the new technology that provides us with the tools to store the summarized information from multiple, heterogeneous databases in a single repository. Data warehousing is the process of integrating enterprise-wide corporate data into a single repository, from which end-users can easily run queries, make reports and perform analysis. A data warehouse is a decision-support environment that leverages data stored in different sources, organizing it and delivering it to decision makers across the enterprise, regardless of their platform or technical skill level. In a nutshell,

data warehousing is the data management and analysis technology adopting an update-driven principle. Therefore, a large number of organizations have found that data warehouse systems are valuable tools in today's competitive and fast evolving world.

The data warehouse is a new approach to enterprise-wide computing at the strategic or architectural level. A data warehouse can provide a central repository for large amounts of diverse and valuable information. By filing the data into a central point of storage, the data warehouse provides an integrated representation of the multiple sources of information dispatch across the enterprise. It ensures the consistency of management rules and conventions applied to the data. It also provides the appropriate tools to extract specific data, convert it into business information, and monitor for changes and, hence, it is possible to use this information to make insightful decisions. A data warehouse is a competitive tool that gives every end user the ability to access quality enterprise-wide data.

In this chapter, we shall briefly study warehousing technology. In Section 2.2, we discuss the essential feature of a data warehouse. In Section 2.3, we present the formal definition of data warehousing and discuss each aspect of the definition. Section 2.4 deals with multidimensional database models. In this section, we study the data cube concepts of warehousing. OLAP is one of the important components of warehousing architecture. We study OLAP and related concepts in Section 2.5. In Section 2.6, different schemas required for data warehousing design are discussed. In Section 2.5, we study the different logical designs of the warehouse. The architecture of the warehouse is given in Section 2.7. The subsequent sections (Section 2.8–2.11) elaborate the different components of a warehouse. At the end of the chapter, there are some exercises.

2.2 What is a Data Warehouse?

Let us first ask ourselves, "What is a Data Warehouse?". From the foregoing discussion, we can infer that a data warehouse supports business analysis and decision making by creating an enterprise-wide integrated database of summarized, historical information. It integrates data from multiple, incompatible sources. By transforming data into meaningful information, a data warehouse allows the business manager to perform more substantive, accurate and consistent analysis. Data warehousing improves the productivity of corporate decision-makers through consolidation, conversion, transformation and integration of operational data and provides a consistent view of the enterprise. Let us try to understand the technology involved in the data warehousing process.

After knowing this much about a data warehouse, a natural question that comes to our minds is: "How is a data warehouse different from a database?". A data warehouse is supposed to be a place where data gets stored so that applications can access and

share it easily. But a database does that already. So then, what makes a warehouse so different?

To answer this question, we can say that a data warehouse is a database of different kind. It is not the normal database, as we understand the term "database". The main difference is that *usual* (or, *traditional)* databases hold operational-type (most often, transactional-type) data and that many of the decision-support type applications put too much strain on the databases intervening into the day-to-day operation. A data warehouse is of course a database, but it contains summarized information. In general, our database is not a data warehouse unless we also

- collect and summarize information from disparate sources and use it as the place where this disparity can be reconciled, and
- place the data into a warehouse because we intend to allow several different applications to make use of the same information.

Loosely speaking, a data warehouse refers to a database that is maintained separately from an organization's operational databases. An operational database is designed and tuned for known tasks and workloads, such as indexing and hashing using primary keys, searching for particular records and optimized "canned" queries. On the other hand, data warehouse queries are often very complex. They involve the computation of large groups of data at the summarized level, and may require the use of special data organizations, access and implementations methods based on multidimensional views.

Another important criterion that almost every one seems to agree upon for a definition is that a warehouse holds *read-only* data. These criteria bring the term, data warehouse, much closer to our understanding of a warehouse as a place where we store many different things for the sake of convenience.

In the following section, we shall discuss the formal definition of data warehousing.

2.3 Definition

Let us now study the formal definition of a data warehouse. W H Inmon (1993) offers the following definition of a data warehouse.

*A data warehouse is a **subject-oriented, integrated, time-varying, non-volatile** collection of data in support of the management's decision-making process.*

Subject-oriented

A data warehouse is organized around major subjects such as customer, products, sales, etc. Data are organized according to subject instead of application. For example,

an insurance company using a data warehouse would organize their data by customer, premium, and claim instead of by different products (auto, life, etc.). The data organized by subject obtains only the information necessary for the decision support processing.

NON-VOLATILE

A data warehouse is always a physically separate store of data, which is transformed from the application data found in the appropriate environment. Due to this separation, data warehouses do not require transaction processing, recovery, concurrency control, etc. The data are not updated or changed in any way once they enter the data warehouse, but are only loaded, refreshed and accessed for queries.

TIME-VARYING

Data are stored in a data warehouse to provide a historical perspective. Every key structure in the data warehouse contains, implicitly or explicitly, an element of time. The data warehouse contains a place for sorting data that are 5 to 10 years old, or older, to be used for comparisons, trends and forecasting.

INTEGRATED

A data warehouse is usually constructed by integrating multiple, heterogeneous sources such as relational databases, flat files, and OLTP files. When data resides in many separate applications in the operational environment, the encoding of data is often inconsistent. When data are moved from operational environment into the data warehouse, they assume a consistent coding convention. Data cleaning and data-integration techniques are applied to maintain consistency in naming convention, measures of variables, encoding structure, and physical attributes.

Let us now investigate the technology underlying the process of data warehousing. In the following section we study the most important concept of data warehousing, that is, the Multidimensional Data Model.

2.4 MULTIDIMENSIONAL DATA MODEL

At the core of the design of the data warehouse lies a multidimensional view of the data model. In order to understand this concept, consider the way the statistical tables are traditionally represented.

Study the data set represented in the following Table 2.1 as a 2-D table, (adopted from [S Choudhury, 1997]). It shows *Employment in California* by *sex*, by *year* and by

Table 2.1 Statistical Table: Two-dimensional representation

			Professional Class					
			Engineer		Secretary		Teaching	
			PROFESSION		PROFESSION		PROFESSION	
			Chemical Engineers	Civil Engineers	Junior Secretary	Executive Secretary	Elementary Teachers	High School Teachers
SEX	MALE	91	1977	2411	5343	1541	2129	1237
		92	2099	2780	5421	1698	2135	1457
		93	2237	3352	5862	1854	2211	1583
		94	2354	3882	5461	1512	2121	1548
		95	2078	3282	5664	1711	2053	1380
	FEMALE	91	258	1120	6673	1623	2160	2751
		92	289	1276	6925	1744	2175	2993
		93	312	1398	7152	1889	2189	3125
		94	581	1216	6543	1534	2857	2387
		95	329	.1321	6129	1567	2453	3287

profession. This form of representing multidimensional tables is very popular in Statistical Data Analyses, because in the early days it was only possible to represent information on paper and thus the 2-D restriction. We observe that the rows and the columns represent more than one dimension, if the data set contains more than 2 dimensions. Selecting an arbitrary order of the dimensions for the rows and the columns does this. The rows in Table 2.1 represent the two dimensions; *sex* and *year*, which are ordered as sex first then year (the order is arbitrary). The columns, however, do not really represent 2 distinct dimensions but they do represent some sort of taxonomy of a dimension. The professional class and profession represent a hierarchical relationship between the instances of professional class and the instances of the profession. We also observe that there is summary information, which is the main theme of the table. In this case the summary function is **sum**.

Data Cube

A popular conceptual model that influences data warehouse architecture is a multidimensional view of data. Figure 2.1 demonstrates a multidimensional view of the information corresponding to Table 2.1. This model views data in the form of a data cube (or, more precisely, a hypercube). We shall use the term *data cube*. It has three dimensions, namely *sex, profession* and *year*. Each dimension can be divided into subdimensions.

In a multidimensional data model, there is a set of numeric measures that are the main theme or subject of the analysis. In the above example, the numeric measure is

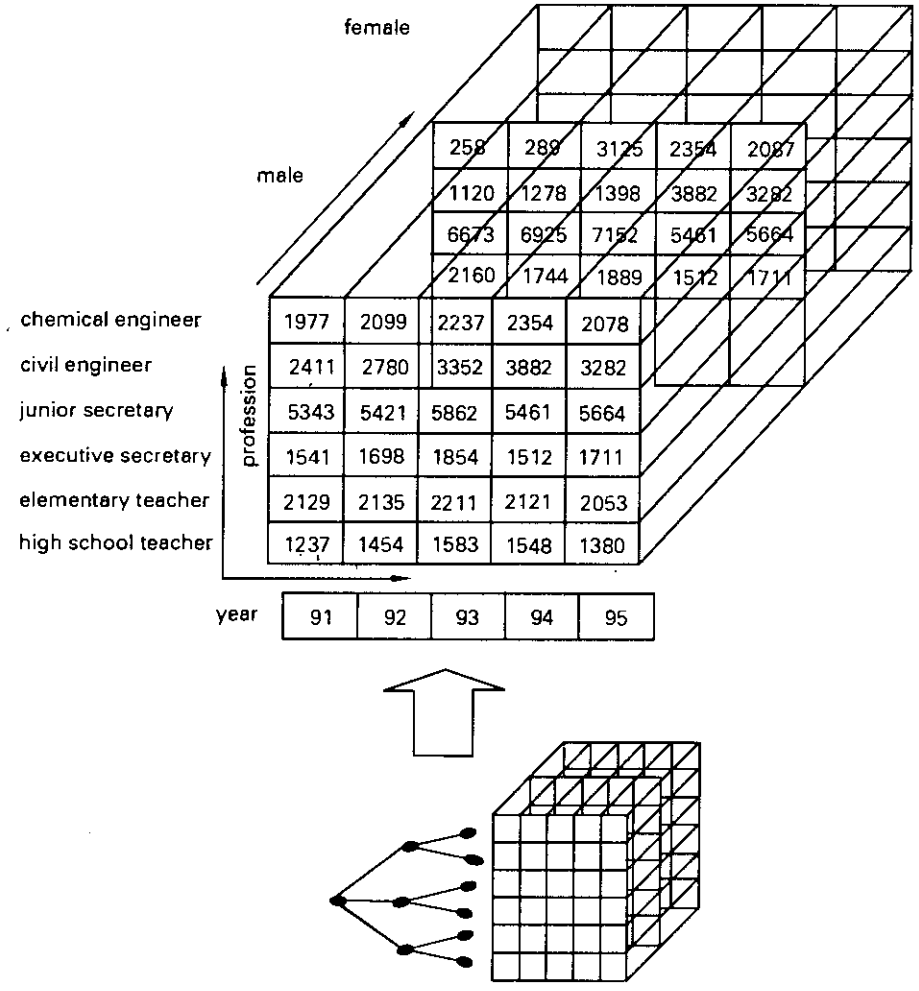


Figure 2.1 Multidimensional Representation of Data

employment. We can have more than one numeric measure. Some examples of numeric measures are sales, budget, revenue, inventory, population, etc. Each numeric measure depends on a set of dimensions, which provide the context for the measure. All the dimensions together are assumed to uniquely determine the measure. Thus, the multidimensional data views a measure as a value placed in a cell in the multidimensional space. Each dimension, in turn, is described by a set of attributes. In general terms, dimensions are the perspectives or entities with respect to which an organization wants to keep records. Each dimension is described by a set of attributes. The attributes of a dimension may be related via a hierarchy of relationships or by a lattice. We can now formally define a data cube at this stage.

An n-dimensional data cube, $C[A_1, A_2, ..., A_n]$, is a database with n dimensions as $A_1, A_2, ...,$ and A_n, each of which represents a theme and contains $|A_i|$ number of distinct elements in the dimension A_i. Each distinct element of A_i corresponds to a data row of C. A data cell in the cube, $C[a_1, a_2, ..., a_n]$ stores the numeric measures of the data for $A_i = a_i$, $\forall i$. Thus, a data cell corresponds to an instantiation of all dimensions.

In the above example, **C[sex, profession, year]** is the data cube, and a data cell **C[male, civil engineer, 1992]** stores 2780 as its associated measure. As $|sex| = 2$, $|profession| = 6$ and $|year| = 5$, we have three dimensions with 2, 6 and 5 rows, respectively.

Dimension Modelling

The notion of a dimension provides a lot of semantic information, especially about the hierarchical relationship between its elements. It is important to note that dimension modelling is a special technique for structuring data around business concepts. Unlike ER modelling, which describes entities and relationships, dimension modelling structures the numeric measures and the dimensions. The dimension schema can represent the details of the dimensional modelling. The following figures show the dimension modelling for our example.

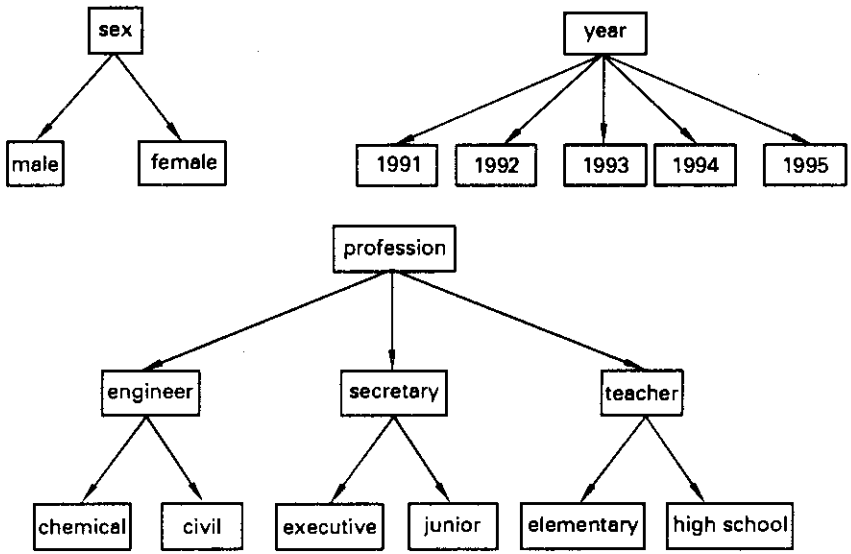


Figure 2.2 Dimension Modelling

Lattice of Cuboids

The dimension hierarchy helps us view the multidimensional data in several different data cube representations. Conceptually, multidimensional data can be viewed as a

lattice of cuboids. The $C[A_1, A_2, \ldots, A_n]$ at the finest level of granularity is called the *base cuboid* and it consists of all the data cells. The $(n-1)$-D cubes are obtained by grouping the cells and computing the combined numeric measure of a given dimension. Finally, the coarsest level consists of one cell with numeric measures of all n dimensions. This is called an *apex cuboid*. In the lattice of cuboids, all other cuboids lie between the base cuboid and the apex cuboid. For our example, the lattice of cuboids is a trivial one and it contains just two cuboids—the base cuboid and the apex cuboid. We shall study a better example later.

Thus, we observe that a multidimensional data model has the following conceptual basic components:

1. **Summary measure**: e.g., employment, sales, etc.
2. **Summary function**: e.g., sum
3. **Dimension**: e.g., sex, year, profession, state
4. **Dimension hierarchy**: e.g., professional class → profession.

Summary Measures

As we have noted, the summary measure is essentially the main theme of the analysis in a multidimensional model. A measure value is computed for a given cell by aggregating the data corresponding to the respective dimension-value sets defining the cell. The measures can be categorized into 3 groups based on the kind of aggregate function used.

Distributive A numeric measure is distributive if it can be computed in a distributed manner as follows. Suppose the data is partitioned into a few subsets. The measure can be simply the aggregation of the measures of all partitions. For example, *count, sum, min* and *max* are distributive measures.

Algebraic An aggregate function is algebraic if it can be computed by an algebraic function with some set of arguments, each of which may be obtained by a distributive measure. For example, *average* is obtained by sum/count.

Other examples of distributive functions are *standard-deviations* and *center-of-mass*.

Holistic An aggregate function is holistic if there is no constant bound on the storage size needed to describe a subaggregate. That is, there does not exist an algebraic function that can be used to compute this function. Examples of such functions, are *median, mode, most-frequent*, etc.

Example 2.2

Let us consider another example at this stage. A store called, *Deccan Electronics* may

create a sales data warehouse in order to keep records of the store's sales with respect to the time, product and location. Thus, the dimensions are *time, product* and *location*. These dimensions allow the store to keep track of things like monthly sales of items, and the locations at which the items were sold. A dimension table for *product* contains the attributes *item name, brand*, and *type*. The attributes *shop, manager, city, region, state,* and *country* describe the dimension *location*. These attributes are related by a total order forming a hierarchy, such as *shop* < *city* < *state* < *country*. This hierarchy is shown in Figure 2.3. An example of a partial order for the *time* dimension are the attributes *week, month, quarter* and *year*. The sales data warehouse includes the sales amount in rupees and the total number of units sold. Note that we can have more than one numeric measure. Figure 2.4 shows the multidimensional model for such

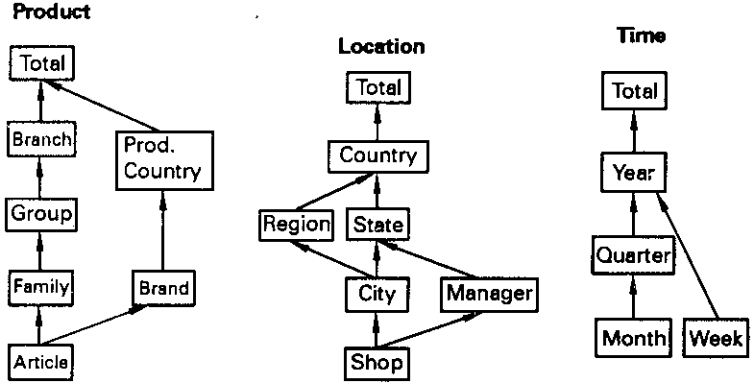


Figure 2.3 Dimension Schema

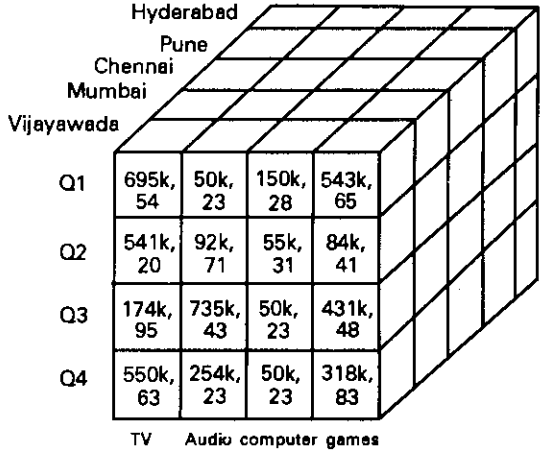


Figure 2.4 Data Cube for Example 2

situations. For convenience, we do not take all the subdivisions of the dimensions into account. The dimension hierarchies considered for the data cube are *time: (month < quarter < year); location: (city < province < country); and product.*

Figure 2.4 shows the cuboid ***C[quarter, city, product]***. In the given dimensional hierarchies, the base cuboid of the lattice corresponds to ***C[month, city, product]*** and the apex cuboid is ***C[year, country, product]***. Other intermediate cuboids in the lattice are ***C[quarter, province, product]***, ***C[quarter, country, product]***, ***C[month, province, product]***, ***C[month, country, product]***, ***C[year, city, product]*** and ***C[year, province, product]***.

2.5 OLAP OPERATIONS

Once we model our data warehouse in the form of a multidimensional data cube, it is necessary to explore the different analytical tools with which to perform the complex analysis of data. These data analysis tools are called OLAP (On-Line Analytical Processing). OLAP is mainly used to access the live data online and to analyze it. OLAP tools are designed in order to accomplish such analyses on very large databases. Hence, OLAP provides a user-friendly environment for interactive data analysis. Whilst data warehousing is primarily targeted at providing consolidated data for further analysis, OLAP provides the means to analyze those data in an application-oriented manner.

In the multidimensional model, the data are organized into multiple dimensions and each dimension contains multiple levels of abstraction. Such an organization provides the users with the flexibility to view data from different perspectives. There exist a number of OLAP operations on data cubes which allow interactive querying and analysis of the data. According to the underlying multidimensional view with classification hierarchies defined upon the dimensions, OLAP systems provide specialized data analysis methods. In this section, we study the basic OLAP operations for a multidimensional model.

Slicing This operation and the following one, Dicing, are used for reducing the data cube by one or more dimensions. The slice operation performs a selection on one dimension of the given cube, resulting in a subcube. Figure 2.5 shows a slice operation where the sales data are selected from the central cube for the dimension time, using the criteria *time*= '*Q2*'.

slice $_{\text{time='Q2'}}$ ***C*[quarter, city, product] = *C*[city, product]**

Dicing This operation is for selecting a smaller data cube and analyzing it from different perspectives. The dice operation defines a subcube by performing a selection on two or more dimensions. Figure 2.6 shows a dice operation on the central cube based on the following selection criteria, which involves three dimensions: *(location= "Mumbai" or "Pune")* and (time = "*Q*1" or "*Q2*").

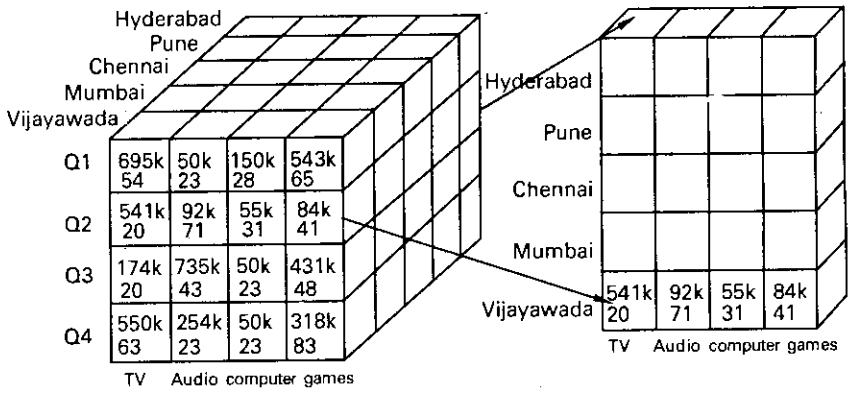


Figure 2.5 Slice Operation

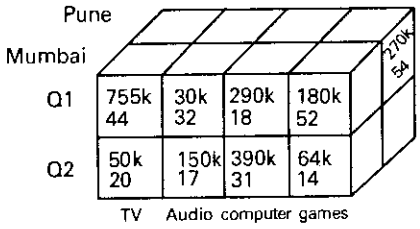


Figure 2.6 Dice Operation

$$\textbf{dice}_{\text{time='Q1' or 'Q2' and location = "Mumbai" or "Pune"}}\ \boldsymbol{C}[\textbf{quarter, city, product}]$$
$$= \boldsymbol{C}[\textbf{quarter', city', product}].$$

Where **quarter' and city'** have truncated domains such as {Q1, Q2} and {Mumbai, Pune}, respectively.

Drilling This operation is meant for moving up and down along classification hierarchies. The different instances of drilling operations may be distinguished as follows:

Drill-up This operation deals with switching from a detailed to an aggregated level within the same classification hierarchy. The drill-up operation (also called as **roll-up** operation) performs aggregation on a data cube, either by *climbing-up a dimension hierarchy* or by *dimension* reduction. Figure 2.7 shows the result of a roll-up operation performed on the central cube by climbing up the dimension hierarchy for the *location* given in Figure 2.3. The roll-up operation aggregates the data by ascending the location hierarchy from the level of the city to the level of *province.* In other words, rather than grouping the data by the city, the resulting cube groups the data by the province.

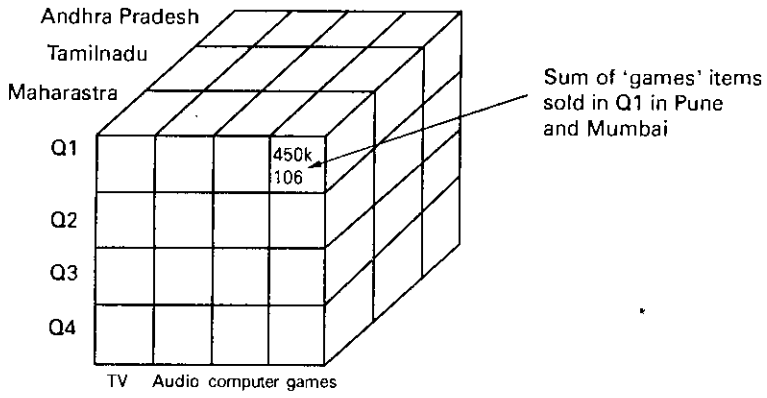


Figure 2.7 Roll-up Operation

roll-up$_{time}$ ***C*[quarter, city, product] = *C*[quarter, province, product]**

Here, each data cell of the resulting cuboid is the aggregation of the data cells that are merged due to the roll-up operation. In other words, the measures stored in the data cells, ***C*[Q1, Mumbai, computer]** and ***C*[Q1, Pune, computer]**, are added to determine the measure to be stored at ***C*[Q1, Maharashtra, computer]** (Maharashtra is a province or state in India and the cities Mumbai and Pune are in this province).

When roll-up is performed by dimension reduction, one or more dimensions are removed from the given cube. For example, consider a employment data cube containing only the two dimensions, profession and time. Roll-up may be performed by removing, say, the sex dimension, resulting in an aggregation of the total employment by profession and by year.

Drill-down This operation is concerned with switching from an aggregated to a more detailed level within the same classification hierarchy. Drill-down is the reverse

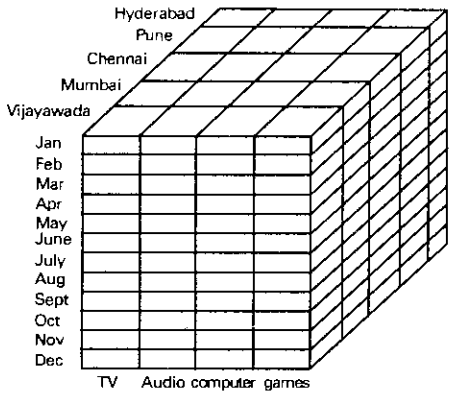


Figure 2.8 Drill-down Operation

of roll-up. It navigates from less detailed data to more detailed data. Drill-down can be realized by either *stepping-down* a dimension hierarchy or *introducing additional dimensions.*

Figure 2.8 shows the result of a drill-down operation performed on the central cube by stepping down a dimension hierarchy for a time period defined as *day* < *month* < *quarter* < *year*. Drill-down occurs by descending in the time hierarchy from the level of a quarter to the level of a month. The resulting data cube details the total sales per month rather than summarized by quarter. Drill-down can also be performed by adding new dimensions to a cube. For example, a drill-down on the given cube can occur by introducing an additional dimension, such as *customer-type.*

Drill-within It is switching from one classification to a different one within the same dimension;

Drill-across It means switching from a classification in one dimension to a different classification in a different dimension.

Pivot (rotate) Pivot (also called "rotate") is a visualization operation which rotates the data axes in order to provide an alternative presentation of the same data. Other examples include rotating the axes in a 3-D cube, or transforming a 3-D cube into a series of 2-D planes.

Other OLAP operations may include ranking the top-N or bottom-N items in lists, as well as computing moving averages, growth rates, interests, and internal rates of return, depreciation, currency conversions, and statistical functions. The results of these operations are typically visualized in a cross-tabular form, i.e., mapped to a grid-oriented, two-dimensional layout structure.

2.6 Warehouse Schema

Having studied data cubes and OLAP operations as the core concepts for data warehouse design, we now ask the question—"How do we go about actually designing the warehouse?" A multidimensional data model identifies the dimensions, their hierarchies, the measure functions, etc. for the design of a data cube. But there are other types of information that are not yet captured. For example, for a particular city, we may like to store its population, type and other attributes which do not participate in any of the hierarchies. The other question is—"How do we actually realize a data cube?" We can make use of multidimensional arrays to define the data cube and the well-known matrix arithmetics for OLAP operations. When we have such a model for OLAP operations, we make use of the MOLAP engine. On the other hand, we can use the existing relational technology and hence, in that case, the engine is a Relational OLAP (ROLAP) for analysis. We shall study

the detailed features of MOLAP and ROLAP in a subsequent section. In this section, we address the problem of designing data warehouse schemas for this purpose.

Star Schema

A *star schema* is a modelling paradigm in which the data warehouse contains a large, single, central *Fact Table* and a set of smaller *Dimension Tables*, one for each dimension. The Fact Table contains the detailed summary data. Its primary key has one key per dimension. Each dimension is a single, highly denormalized table. Every tuple in the Fact Table consists of the *fact or subject* of interest, and the dimensions that provide that fact. Each tuple of the Fact Table consists of a (foreign) key pointing to each of the Dimension Tables that provide its multidimensional coordinates. It also stores numerical values (non-dimensional attributes, and results of statistical functions) for those coordinates. The Dimension Tables consist of columns that correspond to the attributes of the dimension. So each tuple in the Fact Table corresponds to one and only one tuple in each Dimension Table. Whereas, one tuple in a Dimension Table may correspond to more than one tuple in the Fact Table. So we have a 1:N relationship between the Fact Table and the Dimension Table. The advantages of a star schema is that it is easy to understand, easy to define hierarchies, reduces the number of physical joins, requires low maintenance and very simple metadata.

Example 2.3

Let us consider the "Employment" data warehouse. We have three Dimension Tables and one Fact Table. The Star Schema for this example is shown in Figure 2.9.

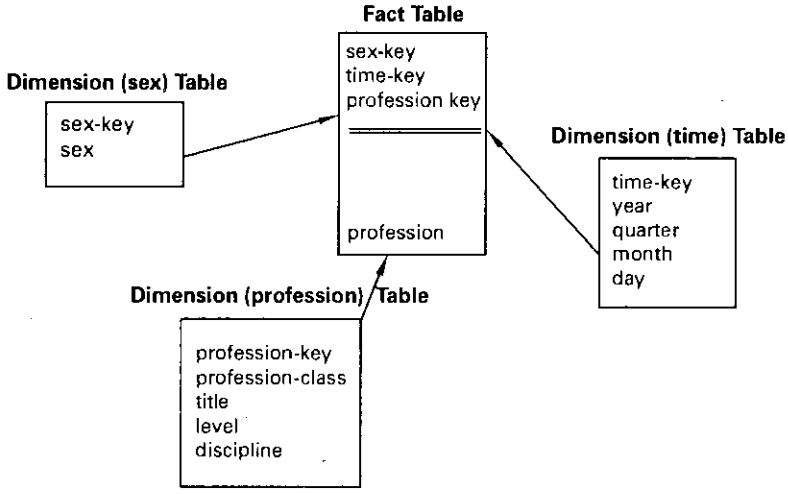


Figure 2.9 Star Schema

SNOWFLAKE SCHEMA

We notice that star schema consists of a single fact table and a single denormalized dimension table for each dimension of the multidimensional data model. To support attribute hierarchies, the dimension tables can be normalized to create *snowflake schemas*. A snowflake schema consists of a single fact table and multiple dimension tables. Like the Star Schema, each tuple of the fact table consists of a (foreign) key pointing to each of the dimension tables that provide its multidimensional coordinates. It also stores numerical values (non-dimensional attributes, and results of statistical functions) for those coordinates. Dimension Tables in a *star* schema are denormalized, while those in a *snowflake* schema are normalized. A generalized view of the snowflake schema is presented in Figure 2.10. The advantage of the

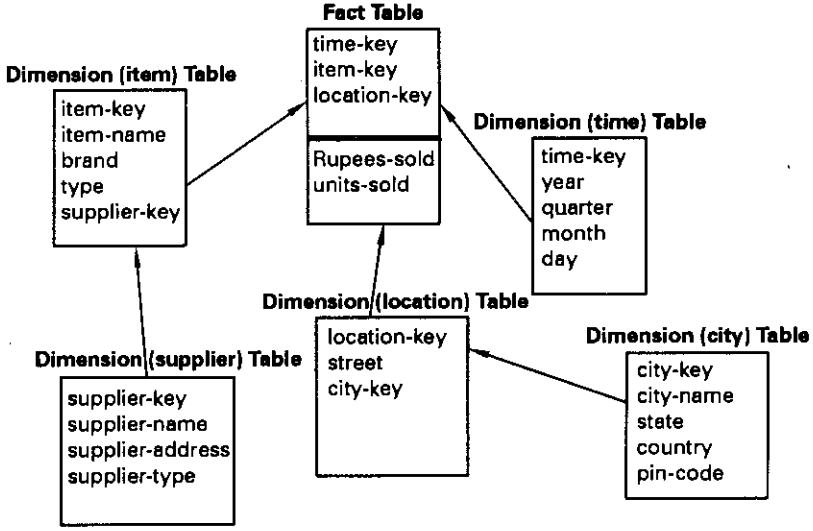


Figure 2.10 Snowflake Schema

snowflake schema is as follows. A normalized table is easier to maintain. Normalizing also saves storage space, since an un-normalized Dimension Table tends to be large and may contain redundant information. However, the snowflake structure may be reducing the effectiveness of navigating across the tables due to a larger number of join operations.

EXAMPLE 2.4

An example of a snowflake schema for a company Deccan Electronics is given in Figure 2.10. It can be seen that the dimension table for the items is normalized resulting in two tables namely, the item and supplier tables.

FACT CONSTELLATION

Most often, there may be a need to have more than one Fact Table and these are called Fact constellations. A *Fact Constellation* is a kind of schema where we have more than one Fact Table sharing among them some Dimension Tables. It is also called *Galaxy Schema*. For example, let us assume that Deccan Electronics would like to have another Fact Table for supply and delivery. It may contain five dimensions, or keys: time, item, delivery-agent, origin, destination along with the numeric measure as the number of units supplied and the cost of delivery. It can be seen that both Fact Tables can share the same item-Dimension Table as well as time-Dimension Table.

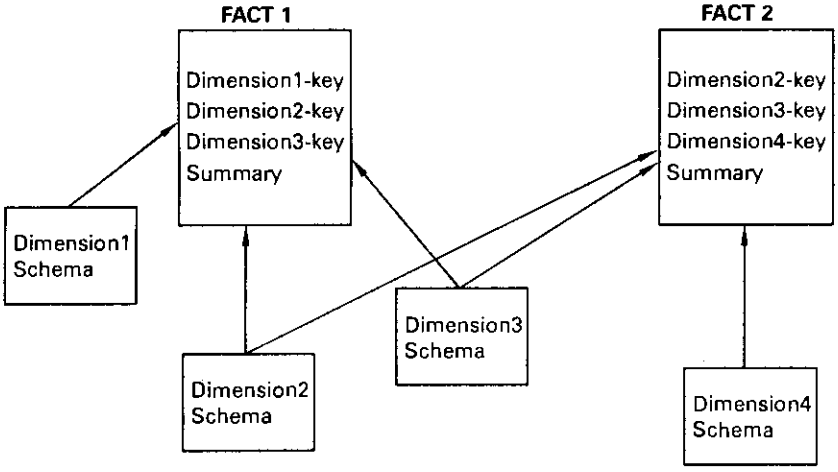


Figure 2.11 Fact Constellation: Fact1 and Fact2 Share the same Dimension Tables, Dim2 and Dim3

2.7 DATA WAREHOUSING ARCHITECTURE

Based on the logical data model of the data warehouse, let us now study the architecture of the warehouse. In this section, we shall study the popular 3-tier architecture and the components of the warehouse at different layers. In the following sections, we discuss the features of these components in detail.

Figure 2.12 shows a typical data warehousing architecture. Very often, this structure can be visualized as 3-tier architecture. Tier 1 is essentially the warehouse server, Tier 2 is the OLAP-engine for analytical processing, and Tier 3 is a client containing reporting tools, visualization tools, data mining tools, querying tools, etc. There is also the backend process which is concerned with extracting data from multiple operational databases and from external sources; with cleaning, transforming and integrating this data for loading into the data warehouse server; and of course, with periodically refreshing the warehouse. Tier 1 contains the main data warehouse. It can follow one

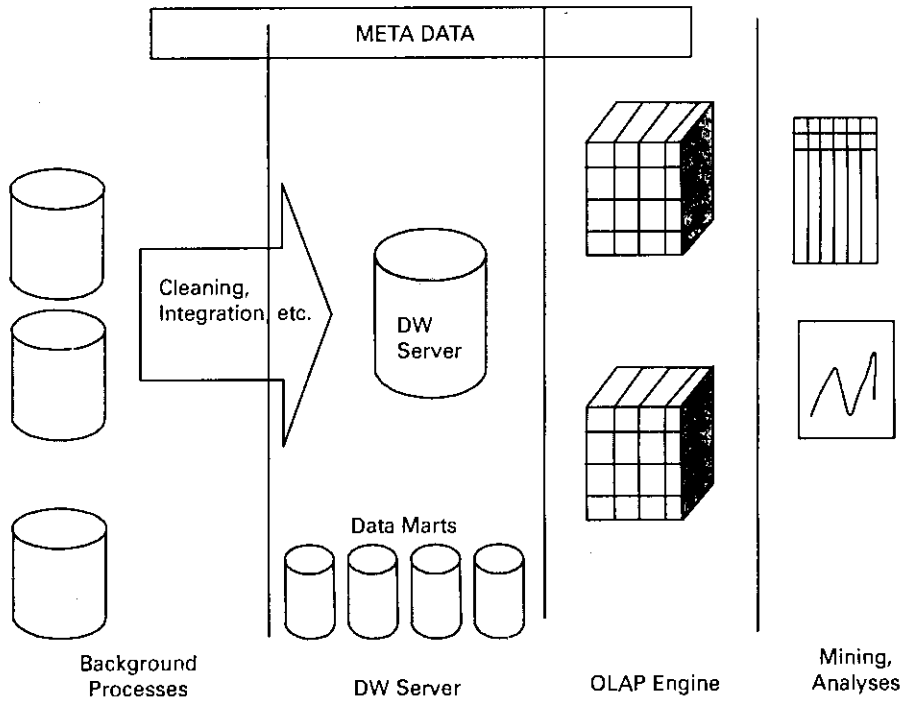


Figure 2.12 Data Warehouse Architecture

of three models or some combination of these. It can be single enterprise warehouse, or may contain several departmental marts. The third model is to have a virtual warehouse. We shall explain these concepts in the following sections. Tier 2 follows three different ways of designing the OLAP engine, namely ROLAP, MOLAP and extended SQL OLAP.

2.8 Warehouse Server

The warehouse server sits at the core of the architecture described above. We shall discuss different models of the warehouse server. As mentioned earlier, there are three data warehouse models.

Enterprise Warehouse

This model collects all the information about the subjects, spanning the entire organization. It provides corporate-wide data integration, usually from one or more operational systems or external information providers. An enterprise data warehouse requires a traditional mainframe.

Data Marts

Data Marts are partitions of the overall data warehouse. If we visualize the data warehouse as covering every aspect of a company's business (sales, purchasing, payroll, and so forth), then a data mart is a subset of that huge data warehouse built specifically for a department. Data marts may contain some overlapping data. A store sales data mart, for example, would also need some data from inventory and payroll. There are several ways to partition the data, such as by business function or geographic region.

Historically, the implementation of a data warehouse has been limited to the resource constraints and priorities of the MIS organization. The task of implementing a data warehouse can be a very big effort, taking a significant amount of time. And, depending on the implementing alternatives chosen, this could dramatically impact the time it takes to see a payback or return on investment. There are many alternatives to design a data warehouse. One feasible option is to start with a set of data marts for each of the component departments. One can have a stand-alone data mart or a dependent data mart.

The current trend is to define the data warehouse as a conceptual environment. The industry is moving away from a single, physical data warehouse toward a set of smaller, more manageable, databases called data marts. The physical data marts together serve as the conceptual data warehouse. These marts must provide the easiest possible access to information required by its user community.

Stand-alone Data Mart

This approach enables a department or work-group to implement a data mart with minimal or no impact on the enterprise's operational database.

Dependent Data Mart

This approach is similar to the stand-alone data mart, except that management of the data sources by the enterprise database is required. These data sources include operational databases and external sources of data.

Virtual Data Warehouse

This model creates a virtual view of databases, allowing the creation of a "virtual warehouse" as opposed to a physical warehouse. In a virtual warehouse, you have a logical description of all the databases and their structures, and individuals who want to get information from those databases do not have to know anything about them.

This approach creates a single "virtual database" from all the data resources. The data resources can be local or remote. In this type of a data warehouse, the data is not moved from the sources. Instead, the users are given direct access to the data. The direct access to the data is sometimes through simple SQL queries, view definition, or data-access middleware. With this approach, it is possible to access

remote data sources including major RDBMSs.

The virtual data warehouse scheme lets a client application access data distributed across multiple data sources through a single SQL statement, a single interface. All data sources are accessed as though they are local users and their applications do not even need to know the physical location of the data.

There is a great benefit in starting with a virtual warehouse, since many organizations do not want to replicate information in the physical data warehouse. Some organizations decide to provide both by creating a data warehouse containing summary-level data with access to legacy data for transaction details.

A virtual database is easy and fast, but it is not without problems. Since the queries must compete with the production data transactions, its performance can be considerably degraded. Since there is no metadata, no summary data or history, all the queries must be repeated, creating an additional burden on the system. Above all, there is no clearing or refreshing process involved, causing the queries to become very complex.

2.9 METADATA

Metadata is to the data warehouse what the card catalogue is to the traditional library. It serves to identify the contents and location of data in the warehouse. Metadata is a bridge between the data warehouse and the decision support application. In addition to providing a logical linkage between data and application, metadata can pinpoint access to information across the entire data warehouse, and can enable the development of applications which automatically update themselves to reflect data warehouse content changes.

In a traditional database a schema describes the conceptual or logical data structures of all the objects or entities with which the database is concerned, together with all relationships between them known to the database. In such a well-defined concept the difference between metadata and data disappears—metadata is simple data. However, in the context of the data warehouse, metadata is needed to provide an unambiguous interpretation. Metadata provides a catalogue of data in the data warehouse and the pointers to this data. In addition to this, metadata may contain: data extraction/transformation history, column aliases, data warehouse table sizes, data communication/modelling algorithms and data usage statistics. Metadata is also used to describe many aspects of the applications, including hierarchical relationships, stored formulae, whether calculations have to be performed before or after consolidation, currency conversion information, time series information, item description and notes for reporting, security and access controls, data update status, formatting information, data sources, availability of pre-calculated summary tables, and data storage parameters. In the absence of this information, the actual data is not intelligible and it would not be wise to attempt to view or update it.

A deeper look at what metadata is – expanding the basic "data about data" definition – reveals the following, metadata, in its broadest sense, defines and describes the entire application environment. It answers such questions as

- What does this field mean in business terms?
- Which business processes does this set of queries support?
- When did the job to update the customer data in our data mart last run?
- Which file contains the product data, where does it reside, and what is its detailed structure?

A metadata repository should contain:

- A description of the structure of the data warehouse. This includes the warehouse schema, view, dimensions, hierarchies and derived data definitions, data marts location and contents etc.
- Operational metadata, such as data linkages, currency of data and monitoring information (warehouse usage statistics, error reports, and its trails).
- The summarization processes which include dimension definition, data on granularity, partitions, summary measure, aggregation, summarization etc.
- Details of data sources which include source databases and their contents, gateway descriptions, a data partitions, data extractions, clearing, transformation rules and defaults.
- Data related to system performance, which include indices and profiles that improve data access and retrieval performances, in addition to rules for timing and scheduling of refresh, update, and replication cycles; and
- Business metadata, which includes business terms and definitions, data ownership information, and changing policies.

Types of Metadata

Due to the variety of metadata, it is necessary to categorize it into different types, based on how it is used. Thus, there are three broad categories of metadata.

Build-time Metadata

Whenever we design and build a warchouse, the metadata that we generate ean be termed as build-time metadata. This metadata links business and warehouse terminology and describes the data's technical structure. It is the most detailed and exact type of metadata and is used extensively by warehouse designers, developers, and administrators. It is the primary source of most of the metadata used in the warehouse.

Usage Metadata

When the warehouse is in production, usage metadata, which is derived from build-time

metadata, is an important tool for users and data administrators. This metadata is used differently from build-time metadata, and its structure must accommodate this fact.

Control Metadata

The third way metadata is used is, of course, by the databases and other tools to manage their own operations. For example, a DBMS builds an internal representation of the database catalogue for use as a working copy from the build-time catalogue. This representation functions as control metadata. Most control metadata is of interest only to systems programmers. However, one subset which is generated and used by the tools that populate the warehouse, is of considerable interest to users and data warehouse administrators. It provides vital information about the timeliness of warehouse data and helps users track the sequence and timing of warehouse events.

2.10 OLAP Engine

The main functions of the OLAP engine is to present the user a multidimensional view of the data warehouse and to provide tools for OLAP operations. If the warehouse server organizes the data warehouse in the form of multidimensional arrays, then the implementational considerations of the OLAP engine are different from those when the server keeps the warehouse in a relational form. With these considerations in mind, there are three options of the OLAP engine.

Specialized SQL Server

This model assumes that the warehouse organizes data in a relational structure and the engine provides an SQL-like environment for OLAP tools. The main idea is to exploit the capabilities of SQL. We shall see that the standard SQL is not suitable for OLAP operations. However, some researchers, (and some vendors) are attempting to extend the abilities of SQL to provide OLAP operations. This is of relevance when the data warehouse is available in a relational structure.

Relational OLAP(ROLAP)

The ROLAP approach begins with the premise that data does not need to be stored multidimensionally to be viewed multidimensionally. A scalable, parallel, relational database provides the storage and high-speed access to this underlying data. A middle analysis tier provides a multidimensional conceptual view of the data and an extended analytical functionality which are not available in the underlying relational server. The presentation tier delivers the results to the users. ROLAP systems provide the benefit of full analytical functionality, while maintaining the advantage of relational data.

ROLAP depends on a specialized schema design and its technology is limited by its non-integrated, disparate tier architecture. The problem is that the data is physically separated from the analytical processing. For many queries this is not a major problem, but it limits the scope of analysis. Normally, the ROLAP engine formulates optimized SQL statements that it sends to the RDBMS server. It then takes the data back from the server, reintegrates it, and performs further analysis and computation before delivering the finished results to the user.

Two important features of ROLAP are

- Data warehouse and relational database are inseparable
- Any change in the dimensional structure requires a physical reorganization of the database, which is too time consuming. Certain applications are too fluid for this and the on-the-fly dimensional view of a ROLAP tool is the only appropriate choice.

Multidimensional OLAP (MOLAP)

The third option is to have a special purpose Multidimensional Data Model for the data warehouse, with a Multidimensional OLAP (MOLAP) server for analysis. The traditional ER model tends to be too complex and difficult to navigate, as the most important data warehouse requirement is to have fewer queries accessing a large number of records.

MOLAP servers support multidimensional views of data through array-based data warehouse servers. They map multidimensional views of a data cube to array structures. The advantage of using a data cube is that it allows fast indexing to precompute summarized data. As with a multidimensional data store, storage utilization is low, and MOLAP is recommended in such cases.

ROLAP vs MOLAP

The following arguments can be given in favour of MOLAP:

1. Relational tables are unnatural for multidimensional data.
2. Multidimensional arrays provide efficiency in storage and operations.
3. There is a mismatch between multidimensional operations and SQL.
4. For ROLAP to achieve efficiency, it has to perform outside current relational systems, which is the same as what MOLAP does.

The following arguments can be given in favour of ROLAP:

1. ROLAP integrates naturally with existing technology and standards.
2. MOLAP does not support *ad hoc* queries effectively, because it is optimized for multidimensional operations.

3. Since data has to be downloaded into MOLAP systems, updating is difficult.
4. The efficiency of ROLAP can be achieved by using techniques such as encoding and compression.
5. ROLAP can readily take advantage of parallel relational technology.

The claim that MOLAP performs better that ROLAP is intuitively believable. In a recent paper, this claim was also substantiated by tests. However, the debate will continue as new compression and encoding methods are applied to ROLAP databases.

2.11 Data Warehouse Backend Process

Data warehouse systems use backend tools and utilities to populate and refresh their data. These tools and facilities include the following functions: (1) *data extraction,* which gathers data from multiple, heterogeneous, and external sources; (2) *data cleaning,* which detects errors in the data and rectifies them when possible; (3) *data transformation*, which converts data from legacy or host format to warehouse format; (4) *load,* which sorts, summarizes, consolidates, computes, views, checks integrity, and builds indices and partitions; and (5) *refresh*, which propagates the updates from the data sources to the warehouse.

Data Extraction

Data extraction is the process of extracting data for the warehouse from various sources. The data may come from a variety of sources, such as

- production data,
- legacy data,
- internal *office systems,*
- *external systems,*
- metadata.

Data Cleaning

Since data warehouses are used for decision making, it is essential that the data in the warehouse be correct. However, since large volumes of data from heterogeneous sources are involved, there is a high probability of errors in the data. Therefore, data cleaning is essential in the construction of quality data warehouses. The data cleaning techniques inelude

- *using transformation* rules, e.g., translating attribute names like 'age' to 'DOB';
- *using domain-specific knowledge*;

- *performing parsing and fuzzy* matching, e.g., for multiple data sources, one can designate a preferred source as a matching standard, and
- auditing, i.e., discovering facts that flag unusual patterns.

It is difficult and costly, however, to clean the data that are entered as a result of poor business practices, such as no clear naming conventions, no consistent data standards, etc.

Data Transformation

The sources of data for datawarehouse are usually heterogeneous. Data transformation is concerned with transforming heterogeneous data to an uniform structure so that the data can be combined and integrated.

Loading

Since a data warehouse integrates time-varying data from multiple sources, the volumes of data to be loaded into a data warehouse can be huge. Moreover, there is usually an insufficient time interval when the warehouse can be taken off-line and when loading data, indices and summary tables need to be rebuilt. A loading system should also allow system administrators to monitor the status, cancel, suspend, resume loading or change the loading rate, and restart loading after failures without any loss of data integrity.

There are different data loading strategies.

- Batch loading.
- Sequential loading.
- Incremental loading.

Refresh

When the source data is updated, we need to update the warehouse. This process is called the refresh function. Determining how frequently to refresh is an important issue. One extreme is to refresh on every update. This is very expensive, however, and is normally only necessary when OLAP queries need the most current data, such as Active Data Warehouse, for example, an up-to-the-minute stock quotation. A more realistic choice is to perform refresh periodically. Refresh policies should be set by the data administrator, based on user needs and data traffic.

2.12 Other Features

There are some recent features of data warehousing, which we shall briefly outline

here. Readers are requested to refer to a complete text book on Data Warehousing to study these features in detail.

Warehouse Management Tools

Data warehouse architecture usually provides a set of management tools which include load manager, warehouse manager, query manager. In addition, a data warehouse must be supported by other management tools like server manager, network manager.

Data Warehouse Usage

Data warehouses and data marts are used in a wide range of applications. Business executives in almost every industry use the data collected, integrated, preprocessed, and stored in data warehouses and data marts, to perform data analysis and make strategic decisions. In many firms, data warehouses are used as an integral part of a plan—*execute-assess* or "closed-loop" feedback system for enterprise management. Data warehouses are used extensively in banking and financial services; consumer goods and retail distribution sectors; and controlled manufacturing, such as demand-based production.

Data warehouse is a process that evolves gradually within an enterprise. Typically, the longer that a data warehouse has been in use, the more it will have evolved. Initially, the data warehouse is mainly used for generating reports and answering predefined queries. Eventually, it is used to analyze summarized and detailed data. In the next phase, the data warehouses are used for strategic purposes, performing multidimensional analysis and sophisticated slice-and-dice operations. Finally, the data warehouse may be employed for knowledge discovery and strategic decision making using data mining tools. In this context, the tools for data warehousing can be categorized into *access and retrieval tools, database reporting tools, data analysis tools, and data mining tools*

The Warehouse Atlas

In the data warehouse atlas, metadata provides a variety of high-level views as starting points of data warehouse exploration for various users. It provides views for executive users, for management, a physical features view for data administrators and key business users, and a searchable, no-nonsense index for everyday users. Just as a world atlas often drills down to provide both topographical and political details, a warehouse atlas provides two detailed views—one for end users and one for builders.

THE MISUNDERSTOOD OLAP ENGINE

There are three fundamental misconceptions about OLAP engines.

- OLAP servers can perform data warehousing functions.
  No. OLAP engines build relational cubes that provide the ability to perform multidimensional analysis on a given data set. They are completely inadequate for many tasks commonly associated with data warehouses, such as historical archiving.
- OLAP engines can cleanse and manipulate data being loaded.
  No. OLAP servers focus on providing multidimensional analysis. Most available products emphasize the OLAP functionality and leave the data preparation to the user.
- OLAP engines store the data in a format open to other tools.
  No. There is nothing "open" about OLAP data stores. In order to perform effective roll-up, drill-down, or data pivoting, OLAP servers store their cubes in proprietary formats, if not proprietary file managers. If other application tools have access to that data, it is simply because they have written custom drivers to accommodate the format. OLAP does not and cannot stand on its own. It is a core component of the broader, overall data warehousing application.

2.13 SUMMARY

In this chapter, a brief introduction to data warehousing techniques is outlined. The discussion includes the basic concepts of data warehousing, its architecture and its components. It also addresses the relationship of data warehousing with OLAP and Multidimensional Data Models. We have discussed the concept Data Marts and of Metadata. The three-level architecture also demonstrates the need for data mining tools in the present context. Based on the implementation details, the different OLAP engines like MOLAP and ROLAP are also discussed in this chapter.

EXERCISES

1. A data warehouse is said to contain a 'time-varying' collection of data because
   a. Its contents vary automatically with time
   b. Its life-span is very limited
   c. It contains historical data
   d. It content has explicit time-stamp.

2. The content of a data warehouse is said to be 'non-volatile', because
   a. It remains the same even after the system crashes
   b. Its life-span is very long
   c. It is a read-only data
   d. It disappears when the system is switched off

3. A data warehouse is said to contain a 'subject-oriented' collection of data because
   a. Its contents have a common theme
   b. It is built for a specific application
   c. It cannot support multiple subjects
   d. It is a generalization of "object-oriented'

4. A data warehouse is an 'integrated' collection of data because
   a. It is a collection of data of different types.
   b. It is a collection of data derived from multiple sources
   c. It is a relational database
   d. It contains summarized data

5. A data warehouse is built on historical data and is not guaranteed to be up-to-date information. True or False?

6. A data warehouse is built as a separate repository of data, different from the operational data of an enterprise because
   a. It is necessary to keep the operational data free of any warehouse operations.
   b. A data warehouse cannot afford to allow corrupted data within it.
   c. A data warehouse contains summarized data whereas the operational database contains transactional data

7. In order to populate the data warehouse which of the following set of operations are appropriate?
   a. Refresh and load
   b. Create and edit
   c. Insert and delete
   d. Query and update

8. Dimension data within a warehouse exhibits one of the following properties:
   a. Dimension data consists of the minor part of the warehouse
   b. The aggregated information is actually dimension data
   c. It contains historical data
   d. Dimension data is the information that is used to analyze the elemental transaction.

9. The 'Slice' operation deals with
   a. Selecting all but one dimension of the data cube.
   b. Merging the cells along one dimension

c. Merging cells of all but one dimension
d. Selecting the cells of any one dimension of the data cube.

Consider the following case

CASE STUDY ONE

Institutional finance for the rural sector consists essentially of loans extended by financial institutions for rural development. The financial institutions are Scheduled Commercial Banks, Regional Rural Bank, Cooperatives, Large-size Adivasis Multipurpose Banks. A data warehouse is built to study the institutional finance of the rural sector. We are interested in finding out the year-wise details of loan disbursements and outstanding loans. It is to study the loan disbursement of the bank—group-wise, purpose-wise, region-wise. The country is categorized into different regions namely, North, North-East, East, Central, West and South. Each region consists of a set of states.

10. Given the information of total loans disbursed (in Rs.) during the period 1991–95 and 1996–2000 for each of the states, if we want to get the total loan disbursement for the period 1991–95, for each region then which of the following is the correct sequence of operations ?
    a. A slice followed by dice
    b. A dice followed by slice
    c. A slice followed by roll-up
    d. A slice followed by drill-down
    e. None of these

11. Given the information of the total loans disbursed (in Rs.) during the period 1991–95 and 1996–2000 for each of the states, if we want to arrive at the total loan disbursement year-wise and region-wise, then which of the following is the correct sequence of operations ?
    a. A roll-up followed by dice
    b. A drill-down followed by slice
    c. A slice followed by roll-up
    d. A roll-up followed by drill-down
    e. None of these

12. Given the information of the total loans disbursed (in Rs.) during the period 1991–95 and 1996–2000 for each of the states, if we want to arrive at the total loan disbursement for the period 1991–95, for two states say, Maharashtra and Bihar, then which of the following is the correct sequence of operations ?
    a. A slice only
    b. A dice followed by slice
    c. A dice only

d. A drill-down followed by slice
e. None of these

13. Given the information of the total loans disbursed (in Rs.) during the period 1991–95 and 1996–2000 for each bank group-wise, (there are three bank groups, namely the SBI and its associates, nationalized banks and cooperative banks), if we want to arrive at the total loan disbursement for the period 1991–95 and 1996–2000 for all the banks taken together, which of the following operations is appropriate?
    a. A slice
    b. Roll-up
    c. Drill-down
    d. Dice
    e. No single operation suffices
14. The 'Dice' operation is concerned with
    a. multiple runs of slice
    b. slice on more than one dimension
    c. selecting certain cells of more than one dimension
    d. two consecutive slice operations in two different dimensions
15. The 'Pivot' is an OLAP operation which
    a. integrates several dimensions
    b. is a visualization operation, rotating the axes for alternative presentation
    c. is not a visualization operation
16. A 'virtual warehouse' is essentially
    a. A traditional relational database providing a multidimensional view through a middleware and it is different from the enterprise's operational database
    b. It is the operational database but provides warehouse facilities through a middleware.
    c. It is the materialized view of the operational and transactional database.
17. 'Roll-up' is an OLAP operation
    a. Which switches from the details to the aggregate level within a classification hierarchy.
    b. Which switches from the details to the aggregate level along many dimensions.
    c. Which switches from the aggregate to the detail level within a classification hierarchy
    d. Which switches from the details to the aggregate levels across many classification hierarchies.
18. A 'drill-down ' operation is concerned with
    a. Which merges cells of two dimensions
    b. Which merges cells of any one dimension based on the characteristics of the dimension

c. Which splits cells of two dimensions
d. Which splits cells of any one dimension based on the characteristics of the dimension

19. A 'drill-through' is an OLAP operation that is derived from drill-down in the following sense
    a. Drill-through navigates along many dimensions whereas drill-down navigates only in one dimension
    b. Drill-through moves both upward and downward, whereas drill-down moves only downward.
    c. Drill-through is a bi-directional approach, whereas drill-down is a uni-directional approach

20. Consider the 3-tier architecture of the data warehouse. The OLAP engine corresponds to
    a. The first layer of the architecture
    b. Second layer
    c. Third layer

21. A data mart
    a. is the analysis unit of the overall architecture of a warehouse.
    b. is a stand-alone of a data warehouse, by itself, of a department of the enterprise.
    c. is an essential component, in the sense that every data warehouse necessarily has several data marts.

22. One of the techniques of implementing the OLAP engine is a 'specialized SQL server'. This server exhibits the following properties:
    a. It assumes that the data warehouse is in a multidimensional model and is implemented in a relational DBMS
    b. It facilitates SQL queries for the data warehouse that is physically organized as a multidimensional model
    c. It facilitates OLAP operations in SQL.
    d. It facilitates OLAP operations in SQL when the data warehouse is organized as relational tables.

23. What is 'ROLAP'?
    a. ROLAP is an OLAP engine for (i) multidimensional models and (ii) SQL based OLAP operations.
    b. ROLAP is an OLAP engine for (i) relational model and (ii) SQL-based OLAP operations.
    c. ROLAP is an OLAP engine for (i) Multidimensional models and (ii) SQL queries, but does not support 'slice' and 'dice' operations.
    d. ROLAP is a set of relational operations equivalent to OLAP operations.

24. What is 'MOLAP'?
    a. MOLAP is an OLAP engine for (i) relational models and (ii) multidimensional OLAP operations.
    b. MOLAP is an OLAP engine for (i) multidimensional models and (ii) SQL-based OLAP operations.
    c. MOLAP is an OLAP engine for (i) multidimensional models and (ii) supports multidimensional OLAP operations.
    d. MOLAP is an ROLAP with a supporting multidimensional model.
25. Which of the following statements are false? MOLAP is preferred to ROLAP
    a. when relational tables are not suitable for warehouse modelling.
    b. because multidimensional arrays provide efficiency in storage and operations.
    c. because the current technology of a relational system does not provide full-fledged OLAP operations.
    d. because indexing and hashing techniques are easy to implement for MOLAP.
26. Consider Case Study One
    If we want to design a star schema, the content of the Fact Table would be
    a. The details of all banks and corresponding loans disbursed
    b. Details of the loan amounts disbursed
    c. Details of banks—groups, regions, years
    d. None of these
27. Consider Case Study One
    If we want to design a star schema for this, there are several dimensions. Identify the ones which are not dimensions.
    a. Bank-groups
    b. Purpose
    c. Year
    d. Region
    e. Loan amount
28. Consider Case Study One
    If we want to design a star schema for this, the content of the join-keys will be
    a. Bank-id, Region-id, Loan-id
    b. Bank-id, loan-id, region-id, year-id, purpose-id
    c. Loan-id
    d. None of these
29. Consider Case Study One
    If we want to design a data warehouse, the attributes of a bank such as its manager, interest rates, and address are stored in
    a. Fact tables
    b. Dimension tables
    c. Metadata

30. Which of the following indexing techniques are appropriate for data warehousing
    a. Hashing on primary keys?
    b. Indexing on foreign keys of the fact table
    c. Bit-map indexing
    d. Join indexing

31. What is bit-map indexing?
    a. It is a method of indexing a relational table, where each tuple is mapped to a binary string.
    b. It is a method of indexing fact tables based on signatures
    c. It is a method of mapping a set of tuples to a bucket, based on a single attribute
    d. It is to code whether a particular value of a selected attribute is present in a tuple or not.

32. ROLAP is preferred over MOLAP when
    a. A data warehouse and relational database are inseparable.
    b. The data warehouse is in relational tables, but no slice and dice operations are required.
    c. The multidimensional model does not support query optimization.
    d. A data warehouse contains many fact tables and many dimension tables.

33. Consider the following case

CASE STUDY TWO

The weather data is stored for different locations in a warehouse. The weather data consists of 'temperature', 'pressure', 'humidity' and 'wind velocity'. The location is defined in terms of 'latitude', 'longitude', 'altitude' and 'time'. Assume that nation() is a function that returns the name of the country for a given latitude and longitude.

- Propose a warehousing model for this case.
- Assume that this information is stored in a data warehouse as a data cube. If we want to extract the average temperature and maximum pressure for all locations for every week, which of the following is the appropriate sequence operations on the multidimensional model?
    a. To compute the average temperature and the highest value of pressure for each week, slice along time to get the values week-wise.
    b. The average temperature and maximum pressure are precomputed while constructing the warehouse. It is to carry out only a slice operation on 'time'.
    c. The average temperature and maximum pressure are precomputed while constructing the warehouse. It is to carry out only a roll-up operation on 'time'

d. The average temperature and maximum pressure are to be computed as in (a.) and to carry out slice and drill-down operation on 'time'.
e. None of these

- Assume that this information is stored in a data warehouse as a data cube, along with the average temperature and maximum pressure for all dimensions. Which of the following is the appropriate sequence operations to get the average temperature year-wise?
  a. Slice on time.
  b. Slice on temperature
  c. Roll-up on time
  d. Roll-up on temperature.
  e. None of these
- We want to build a warehouse using the above information with the numeric measures such as average temperature, maximum pressure, maximum and minimum humidity, and average wind velocity. In a constellation (galaxy) schema, which of the following tables store the above information?
  a. Fact table
  b. Summary table
  c. Dimension table

BIBLIOGRAPHY

Agarwal S., Agarwal R., Deshpande P.M., and Gupta A. On the computation of multidimensional aggregates. *VLDB*, 1998.

Agrawal R, Gupta A, and Sarawagi S. Modelling multidimensional databases. *ICDE*, 1997.

Anahory S, and Murray D. *Data Warehousing in the Real World. A Practical Guide for Building Decision Support Systems*. Addison Wesley Longman, 1997.

Chaudhury S., and Dayal U. *An Overview of Data Warehousing and OLAP Technology. SIGMOD Record*, **26**(1): pp. 65–74, 1997.

Delvin Barry. *Data warehouse: From Architecture to Implementation*. Addison Wesley Longman Inc., 1997.

Gray J., and Chaudhuri S. *et al*. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. *Data Mining and Knowledge Discovery*, **1**, pp. 29–53, 1997.

Han J., and Kamber M. *Data Mining: Concepts & Techniques*, Morgan & Kaufmann, 2000.

Harinarayan V., Rajaraman A., and Ullman J. Implementing data cube efficiently. *ACM SIGMOD'96*, 1996.

Inmon W.H. *Building the Data Warehouse* (Second Edition). John Wiley and Sons, 1996.

Inmon W.H., Welch J.D., and Glassey Katherine L. *Managing the Data Warehouse*. John Wiley and Sons, 1997.

Inmon W.H., Zachman John A, and Geiger J.G. *Data Stores, Data Warehousing and the Zachman Framework*. McGraw Hill Series on Data Warehousing and Data Management, 1997.

Kimball R. *The Data Warehouse Toolkit*. John Wiley and Sons, 1996.

Shosani. OLAP and Statistical Databases: Similarity and Differences. *ACM TODS*, 1997.

Thomsen E. *OLAP Solutions*. John Wiley and Sons, 1997.

Widom J. Research problems in Data Warehousing. *ACM CIKM*, 1995.

3

DATA MINING

3.1 INTRODUCTION

We have seen in the earlier chapter that when an enterprise has made substantial investments in capturing, cleaning, storing and managing data for decision support and has built a data warehouse, it is in a position to generate a range of reporting applications through *ad hoc* query capabilities, visualization and drill-down functionalities via OLAP tools. Data mining complements all of these. It also adds value to existing investments by enabling people who understand the warehousing environment (and others) to exploit data assets and gain deeper insights into key aspects of business performance. For example, a marketing department might have well-established models for modelling customer profiles. Data mining helps the marketing analyst to break free of established customer models and construct detailed customer profiles reflecting actual behaviour in different situations. In a marketing situation, such profiles can be used as a basis for enabling promotion costs to be cut

and response ratios to be increased. The net impact of an effective data mining project can be very high.

In this chapter, we shall introduce the concept of data mining. In a sense, we build up arguments to motivate the readers to undertake a detailed study of data mining techniques. We shall present a number of definitions of data mining and discuss the important features of the subject. We shall also identify the major techniques of data mining and discuss possible areas of data mining applications. Data mining is a component of data warehousing, but it can also be stand-alone process for data analysis, even in the absence of a data warehouse.

In Section 3.2, we shall explain the basic concept of data mining. Different definitions of data mining are given in Section 3.3. We also discuss different aspects of these definitions. Though popularly DM and KDD are often used interchangeably, there are some differences between them. This is highlighted in Section 3.4. The DM process can be coupled with a DBMS in tightly-coupled mode or loosely-coupled mode. The advantages and disadvantages of these modes are discussed in Section 3.5. Section 3.6 is concerned with the relationship of DM with other disciplines in Computer Science. Section 3.7 briefly outlines several data mining techniques. Some of these techniques are elaborated in subsequent chapters. It may be noted that database research is no longer confined only to data repository stored in a DBMS. Unstructured data such as text data, time series data or spatial data have already been identified as different fields of research. In Section 8, we shall discuss the data mining problems for these types of data. In Section 3.9, we shall identify the current trends which have influenced the development of DM techniques. Section 3.10 discusses the issues and trends of DM. There are many possible areas where DM methodology can be applied. In Section 3.10, we identify the sources of voluminous data and hence the possible domains of applications of DM. Section 3.12 illustrates some typical applications of DM.

3.2 What is Data Mining?

Data mining is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data. With the widespread use of databases and the explosive growth in their sizes, organizations are faced with the problem of information overload. The problem of effectively utilizing these massive volumes of data is becoming a major problem for all enterprises. Traditionally, we have been using data for querying a reliable database repository via some well-circumscribed application or canned report-generating utility. While this mode of interaction is satisfactory for a large class of applications, there exist many other applications which demand exploratory data analyses. We have seen that in a data warehouse, the OLAP engine provides an adequate interface for querying summarized and aggregate

information across different dimension-hierarchies. Though such methods are relevant for decision support systems, these lack the exploratory characteristics of querying. The OLAP engine for a data warehouse (and query languages for DBMS) supports query-triggered usage of data, in the sense that the analysis is based on a query posed by a human analyst. On the other hand, data mining techniques support automatic exploration of data. Data mining attempts to source out patterns and trends in the data and infers rules from these patterns. With these rules the user will be able to support, review and examine decisions in some related business or scientific area. This opens up the possibility of a new way of interacting with databases and data warehouses. Consider, for example, a banking application where the manager wants to know whether there is a specific pattern followed by defaulters. It is hard to formulate a SQL query for such information. It is generally accepted that if we know the query precisely we can turn to query language to formulate the query. But if we have some vague idea and we do not know the precisely query, then we can resort to data mining techniques.

The evolution of data mining began when business data was first stored in computers, and technologies were generated to allow users to navigate through the data in real time. Data mining takes this evolutionary process beyond retrospective data access and navigation, to prospective and proactive information delivery. This evolution is due to the support of three technologies that are sufficiently mature: massive data collection, high performance computing and data mining algorithms.

We shall study some definitions of the term data mining in the following section.

3.3 Data Mining: Definitions

Data mining, the extraction of the hidden predictive information from large databases, is a powerful new technology with great potential to analyze important information in the data warehouse. Data mining scours databases for hidden patterns, finding predictive information that experts may miss, as it goes beyond their expectations. When implemented on a high performance client/server or parallel processing computers, data mining tools can analyze massive databases to deliver answers to questions such as which clients are most likely to respond to the next promotional mailing. There is an increasing desire to use this new technology in the new application domain, and a growing perception that these large passive databases can be made into useful actionable information.

Definitions

The term 'data mining' refers to the finding of relevant and useful information from databases. Data mining and knowledge discovery in the databases is a new interdisciplinary field, merging ideas from statistics, machine learning, databases and parallel computing. Researchers have defined the term 'data mining' in many ways.

We discuss a few of these definitions below.

1. *Data mining or knowledge discovery in databases, as it is also known, is the non-trivial extraction of implicit, previously unknown and potentially useful information from the data. This encompasses a number of technical approaches, such as clustering, data summarization, classification, finding dependency networks, analyzing changes, and detecting anomalies.*

Though the terms data mining and KDD are used above synonymously, there are debates on the difference and similarity between data mining and knowledge discovery. In the present book, we shall be using these two terms synonymously. However, we shall also study the aspects in which these two terms are said to be different.

Data retrieval, in its usual sense in database literature, attempts to retrieve data that is stored explicitly in the database and presents it to the user in a way that the user can understand. It does not attempt to extract implicit information. One may argue that if we store 'date-of-birth' as a field in the database and extract 'age' from it, the information received from the database is not explicitly available. But all of us would agree that the information is not 'non-trivial'. On the other hand, if one attempts to find out the average age of the employees in a particular company, it can be visualized as a sort of non-trivial extraction of implicit information. Then, can we say that extracting the average age of the employees of a department from the employees database (which stores the date-of-birth of every employee) is a data-mining task? The task is surely 'non-trivial extraction of implicit information'. It is indeed a type of data mining task, but at a very low level. A higher level task would, for example, be to find correlations between the average age and average income of individuals in an enterprise.

2. *Data mining is the search for the relationships and global patterns that exist in large databases but are hidden among vast amounts of data, such as the relationship between patient data and their medical diagnosis. This relationship represents valuable knowledge about the database, and the objects in the database, if the database is a faithful mirror of the real world registered by the database.*

Consider the employee database and let us assume that we have some tools available with us to determine some relationships between fields, say relationship between age and lunch-patterns. Assume, for example, that we find that most of employees in their thirties like to eat pizzas, burgers or Chinese food during their lunch break. Employees in their forties prefer to carry a home-cooked lunch from their homes. And employees in their fifties take fruits and salads during lunch. If our tool finds this pattern from the database which records the lunch activities of all employees for last few months, then we can term our tool as a data mining tool. The daily lunch

activity of all employees collected over a reasonable period of time makes the database very vast. Just by examining the database, it is impossible to notice any relationship between age and lunch patterns.

3. *Data mining refers to using a variety of techniques to identify nuggets of information or decision-making knowledge in the database and extracting these in such a way that they can be put to use in areas such as decision support, prediction, forecasting and estimation. The data is often voluminous, but it has low value and no direct use can be made of it. It is the hidden information in the data that is useful.*

Data mining is a process of finding value from volume. In any enterprise, the amount of transactional data generated during its day-to-day operations is massive in volume. Although these transactions record every instance of any activity, it is of little use in decision making. Data mining attempts to extract smaller pieces of valuable information from this massive database.

4. *Discovering relations that connect variables in a database is the subject of data mining. The data mining system self-learns from the previous history of the investigated system, formulating and testing hypothesis about rules which systems obey. When concise and valuable knowledge about the system of interest is discovered, it can and should be interpreted into some decision support system, which helps the manager to make wise and informed business decision.*

Data mining is essentially a system that learns from the existing data. One can think of two disciplines which address such problems—Statistics and Machine Learning. Statistics provide sufficient tools for data analysis and machine learning deals with different learning methodologies. While statistical methods are theory-rich-data-poor, data mining is data-rich-theory-poor approach. On the other hand machine learning deals with whole gamut of learning theory, which most often data mining is restricted to areas of learning with partially specified data.

5. *Data mining is the process of discovering meaningful, new correlation patterns and trends by sifting through large amount of data stored in repositories, using pattern recognition techniques as well as statistical and mathematical techniques.*

One important aspect of data mining is that it scans through a large volume of data to discover patterns and correlations between attributes. Thus, though there are techniques like clustering, decision trees, etc., existing in different disciplines, these are not readily applicable to data mining as they are not designed to handle large amounts of data. Thus, in order to apply statistical and mathematical tools, we have to modify these techniques to be able efficiently sift through large amounts of data stored in the secondary memory.

3.4 KDD vs. Data Mining

Knowledge Discovery in Database (KDD) was formalized in 1989, with reference to the general concept of being broad and high level in the pursuit of seeking knowledge from data. The term *data mining* was then coined; this high-level application technique is used to present and analyze data for decision-makers.

Data mining is only one of the many steps involved in knowledge discovery in databases. The various steps in the knowledge discovery process include data selection, data cleaning and preprocessing, data transformation and reduction, data mining algorithm selection and finally the post-processing and the interpretation of the discovered knowledge. The KDD process tends to be highly iterative and interactive. Data mining analysis tends to work up from the data and the best techniques are developed with an orientation towards large volumes of data, making use of as much data as possible to arrive at reliable conclusions and decisions. The analysis process starts with a set of data, and uses a methodology to develop an optimal representation of the structure of data, during which knowledge is acquired. Once knowledge is acquired, this can be extended to large sets of data on the assumption that the large data set has a structure similar to the simple data set.

Fayyad *et al.* distinguish between KDD and data mining by giving the following definitions.

Knowledge Discovery in Databases is the process of identifying a valid, potentially useful and ultimately understandable structure in data. This process involves selecting or sampling data from a data warehouse, cleaning or preprocessing it, transforming or reducing it (if needed), applying a data mining component to produce a structure, and then evaluating the derived structure.

Data Mining is a step in the KDD process concerned with the algorithmic means by which patterns or structures are enumerated from the data under acceptable computational efficiency limitations.

Thus, the structures that are the outcome of the data mining process must meet certain conditions so that these can be considered as knowledge. These conditions are: *validity, understandability, utility, novelty* and *interestingness*.

Stages of KDD

The stages of KDD, starting with the raw data and finishing with the extracted knowledge, are given below.

SELECTION

This stage is concerned with selecting or segmenting the data that are relevant to some criteria. For example, for credit card customer profiling, we extract the type of transactions for each type of customers and we may not be interested in details of the shop where the transaction takes place.

PREPROCESSING

Preprocessing is the data cleaning stage where unnecessary information is removed. For example, it is unnecessary to note the sex of a patient when studying pregnancy! When the data is drawn from several sources, it is possible that the same information is represented in different sources in different formats. This stage reconfigures the data to ensure a consistent format, as there is a possibility of inconsistent formats.

TRANSFORMATION

The data is not merely transferred across, but transformed in order to be suitable for the task of data mining. In this stage, the data is made usable and navigable.

DATA MINING

This stage is concerned with the extraction of patterns from the data.

INTERPRETATION AND EVALUATION

The patterns obtained in the data mining stage are converted into knowledge, which in turn, is used to support decision-making.

DATA VISUALIZATION

Data visualization makes it possible for the analyst to gain a deeper, more intuitive understanding of the data and as such can work well alongside data mining. Data mining allows the analyst to focus on certain patterns and trends and explore them in-depth using visualization. On its own, data visualization can be overwhelmed by the volume of data in a database but in conjunction with data mining can help with exploration.

Data visualization helps users to examine large volumes of data and detect the patterns visually. Visual displays of data such as maps, charts and other graphical representations allow data to be presented compactly to the users. A single graphical

screen can encode as much information as can a far larger number of text screens. For example, if a user wants to find out whether the production problems at a plant are correlated to the location of the plants, the problem locations can be encoded in a special colour, say red, on a map. The user can then discover locations in which the problems are occurring. He may then form a hypothesis about why problems are occurring in those locations, and may verify the hypothesis against the database.

3.5 DBMS vs DM

We have seen that DBMS supports query languages which are useful for query-triggered data exploration, whereas data mining supports automatic data exploration. If we know exactly what information we are seeking, a DBMS query would suffice; whereas if we vaguely know the possible correlations or patterns, then data mining techniques are useful. One of the tasks of data mining is hypothesis testing, wherein we formulate a hypothesis and test it by sifting through the database. This task can be handled by a DBMS query. Thus, in these senses, DBMS supports some primitive data mining tasks.

From the architectural perspective, let us look the way a data mining system makes use of an existing database.

There are three different ways in which data mining systems use a relational DBMS. They may not use it at all, be *loosely coupled* or *tightly coupled.*

A majority of data mining systems do not use any DBMS and have their own memory and storage management. They treat the database simply as a data repository from which data is expected to be downloaded into their own memory structures, before the data mining algorithm starts. The advantage of such an approach is that one can optimize the memory management specific to the data mining algorithm. On the contrary, these systems ignore the field-proven technologies of DBMS, such as recovery, concurrency, etc.

The second approach is to have a *loosely-coupled* DBMS. In this case, DBMS is used only for storage and retrieval of data. For instance, one can use a loosely-coupled SQL to fetch data records as required by the mining algorithm. The front-end of the application is implemented in a host programming language, with embedded SQL statements in it. The applications use a SQL select statement to retrieve the set of records of interest from the database. A loop in the application program copies records in the result set one-by-one from the database address space to the application address space, where computation is performed on them. This loosely-coupled approach does not use the querying capability provided by the DBMS.

In *tightly-coupled* approach, the portions of the application programs are selectively pushed to the database system to perform the necessary computation. Data are stored in the database and all processing is done at the database end. It is different from

bringing the data from the database to the data mining area. On the other hand, the data mining application goes where the data naturally reside. This avoids performance degradation and takes full advantage of database technology. The performance of this approach depends on the way to optimize the data mining process while mapping it to a query. There are two suggested approaches. We can leave the optimization task to a built-in query optimizer of the DBMS or we can have an external optimizer.

3.6 Other Related Areas

Data mining research has drawn on a number of other fields such as machine learning and statistics. We review the relations of data mining with some of the important areas.

Statistics

As we mentioned earlier, statistics is a theory-rich approach for data analysis. Statistics, with its solid theoretical foundation, generates results that can be overwhelming and difficult to interpret. These require user guidance as to where and how to analyze the data. Notwithstanding these, statistics is one of the foundational principles on which data mining technology is built. Statistical analysis systems are used by analysts to detect unusual patterns and explain patterns using statistical models, such as linear models. Statistics have an important role to play and data mining will not replace such analyses, but rather statistics can act upon more directed analyses based on the results of data mining.

Machine Learning

Machine learning is the automation of a learning process and learning is tantamount to the construction of rules based on observations. This is a broad field which includes not only learning from examples, but also reinforcement learning, learning with a teacher, etc. A learning algorithm takes the data set and its accompanying information as the input and returns a statement, e.g., a concept representing the results of learning as output.

Inductive learning, where the system infers knowledge itself from observing its environment, has two main strategies: Supervised Learning and Unsupervised Learning. The model produced by inductive learning methods can be used to predict the outcome of future situations; in other words, not only for states encountered but rather for unseen states that could occur. There can be many possible models from a given set of examples. In such situations, the principle *Occam's Razor* is very appropriate. It states that if there are multiple explanations for a particular phenomena,

it makes sense to choose the simplest because it is more likely to capture the nature of the phenomenon.

Supervised Learning

Supervised learning means learning from examples, where a training set is given which acts as examples for the classes. The system finds a description of each class. Once the description (and hence a classification rule) has been formulated, it is used to predict the class of previously unseen objects. This is similar to discriminate analysis which occurs in statistics.

Unsupervised Learning

Unsupervised learning is learning from observation and discovery. In this mode of learning, there is no training set or prior knowledge of the classes. The system analyzes the given set of data to observe similarities emerging out of the subsets of the data. The outcome is a set of class descriptions, one for each class, discovered in the environment. This is similar to cluster analysis in statistics.

Data mining, and the part of machine learning dealing with learning from examples, overlap in the algorithms used and the problems addressed. Data mining is concerned with finding understandable knowledge, while machine learning is concerned with improving the performance of an intelligent system or agent for problem-solving tasks. When integrating machine learning techniques into database systems, some of the databases require more efficient learning algorithms because realistic databases are normally very large and noisy.

Mathematical Programming

The relationship between mathematical programming and data mining was not so obvious until the pioneering work by O L Mangasarian. Most of the major data mining tasks can be equivalently formulated as problems in mathematical programming for which efficient algorithms are available. It provides a new insight into the problems the data mining. One of the major active research topics in this field is *Support Vector Machines* approach for classification. We shall discuss this approach briefly in the following section.

3.7 DM Techniques

Researchers identify two fundamental goals of data mining: prediction and description. *Prediction* makes use of existing variables in the database in order to

predict unknown or future values of interest, and *description* focuses on finding patterns describing the data and the subsequent presentation for user interpretation. The relative emphasis of both prediction and description differ with respect to the underlying application and the technique. There are several data mining techniques fulfilling these objectives. Some of these are associations, classifications, sequential patterns and clustering. The basic premise of an association is to find all associations, such that the presence of one set of items in a transaction implies the other items. Classification develops profiles of different groups. Sequential patterns identify sequential patterns subject to a user-specified minimum constraint. Clustering segments a database into subsets or clusters.

Another approach of the study of DM techniques is to classify the techniques as

- User-guided or verification-driven data mining, and
- Discovery-driven or automatic discovery of rules

Most of the techniques of data mining have elements of both the models.

Verification Model

In this process of data mining, the user makes a hypothesis and tests the hypothesis on the data to verify its validity. The emphasis is on the user who is responsible for formulating the hypothesis and issuing the query on the data to affirm or negate the hypothesis.

In a supermarket, for example, with a limited budget for a mailing campaign to launch a new product, it is important to identify the section of the population most likely to buy the new product. The user formulates a hypothesis to identify potential customers and their common characteristics. Historical data about transactions and demographic information can then be queried to reveal comparable purchases and the characteristics shared by those purchasers. The whole operation can be repeated by successive refinements of hypotheses until the required limit is reached. The user may come up with a new hypothesis or may refine the existing one and verify it against the database.

Discovery Model

The discovery model differs in its emphasis, in that it is the system automatically discovering important information hidden in the data. The data is sifted in search of frequently occurring patterns, trends and generalizations about the data without intervention or guidance from the user.

The manner in which the rules are discovered depends on the class of the data mining application.

An example of such a model is a supermarket database, which is mined to discover the particular groups of customers to target for a mailing campaign. The data is searched with no hypothesis in mind other than for the system to group the customers according to the common characteristics found.

The typical discovery driven tasks are

- Discovery of association rules
- Discovery of classification rules
- Clustering
- Discovery of frequent episodes
- Deviation detection.

These tasks are of an exploratory nature and cannot be directly handed over to currently available database technology. We shall concentrate on these tasks now.

Discovery of Association Rules

An association rule is an expression of the form $X \Rightarrow Y$, where X and Y are the sets of items. The intuitive meaning of such a rule is that the transaction of the database which contains X tends to contain Y. Given a database, the goal is to discover all the rules that have the support and confidence greater than or equal to the minimum support and confidence, respectively.

Let $L=\{l_1,l_2,...,l_m\}$ be a set of literals called items. Let D, the database, be a set of transactions, where each transaction T is a set of items. T supports an item x, if x is in T. T is said to support a subset of items X, if T supports each item x in X. $X \Rightarrow Y$ holds with *confidence* c, if c% of the transactions in D that support X also support Y. The rule $X \Rightarrow Y$ has *support* s in the transaction set D if s% of the transactions in D support $X \cup Y$. *Support* means how often X and Y occur together as a percentage of the total transactions. *Confidence* measures how much a particular item is dependent on another.

Thus, the association with a very high support and confidence is a pattern that occurs often in the database that should be obvious to the end user. Patterns with extremely low support and confidence should be regarded as of no significance. Only patterns with a combination of intermediate values of confidence and support provide the user with interesting and previously unknown information. We shall study the techniques to discover association rules in Chapter 4.

Clustering

Clustering is a method of grouping data into different groups, so that the data in each group share similar trends and patterns. Clustering constitutes a major class of data

mining algorithms. The algorithm attempts to automatically partition the data space into a set of regions or clusters, to which the examples in the table are assigned, either deterministically or probability-wise. The goal of the process is to identify all sets of similar examples in the data, in some optimal fashion.

Clustering according to similarity is a concept which appears in many disciplines. If a measure of similarity is available, then there are a number of techniques for forming clusters. Another approach is to build set functions that measure some particular property of groups. This latter approach achieves what is known as optimal partitioning.

The objectives of clustering are:

- to uncover natural groupings
- to initiate hypothesis about the data
- to find consistent and valid organization of the data.

A retailer may want to know where similarities exist in his customer base, so that he can create and understand different groups. He can use the existing database of the different customers or, more specifically, different transactions collected over a period of time. The clustering methods will help him in identifying different categories of customers. During the discovery process, the differences between data sets can be discovered in order to separate them into different groups, and similarity between data sets can be used to group similar data together. Chapter 5, we shall discuss in detail about using the clustering algorithm for data mining tasks.

Discovery of Classification Rules

Classification involves finding rules that partition the data into disjoint groups. The input for the classification is the training data set, whose class labels are already known. Classification analyzes the training data set and constructs a model based on the class label, and aims to assign a class label to the future unlabelled records. Since the class field is known, this type of classification is known as supervised learning. A set of classification rules are generated by such a classification process, which can be used to classify future data and develop a better understanding of each class in the database. We can term this as *supervised learning* too.

There are several classification discovery models. They are: the decision trees, neural networks, genetic algorithms and the statistical models like linear/geometric discriminates. The applications include the credit card analysis, banking, medical applications and the like. Consider the following example.

The domestic flights in our country were at one time only operated by Indian Airlines. Recently, many other private airlines began their operations for domestic travel. Some of the customers of Indian Airlines started flying with these private

airlines and, as a result, Indian Airlines lost these customers. Let us assume that Indian Airlines wants to understand why some customers remain loyal while others leave. Ultimately, the airline wants to predict which customers it is most likely to lose to its competitors. Their aim to build a model based on the historical data of loyal customers versus customers who have left. This becomes a classification problem. It is a supervised learning task as the historical data becomes the training set which is used to train the model. The decision tree is the most popular classification technique. In Chapter 6, we shall discuss different methods of decision tree construction.

Frequent Episodes

Frequent episodes are the sequence of events that occur frequently, close to each other and are extracted from the time sequences. How close it has to be to consider it as frequent is domain dependent. This is given by the user as the input and the output are the prediction rules for the time sequences.

Given a set R of event types, an event is a pair (A, t) where $A \in R$ is an event type and t is an integer, we can calculate the occurrence time of the event. An event sequence s of R is a triple (T_S, T_C, S), where $T_S < T_C$ are integers. T_S is the starting time and T_C is the ending time.

$S = \{(A_1, t_1), (A_2, t_2), (A_n, t_n)\}$ is the ordered sequence of events, such that $A_i \in R$ and $T_S \leq t_i \leq T_C$ for all $i = 1, 2 \ldots n$–1.

These episodes can be of three types. The serial episodes which occur in sequence. The parallel episodes in which there are no constraints on the order of the event types A and B given. And the non-serial and non-parallel episodes which occur in a sequence if the occurrences of A and B precede an occurrence of C, and there is no constraint on the relative order of A and B given.

The applications include telecommunications, and share market analysis, and these are mainly used for temporal data.

Deviation Detection

Deviation detection is to identify outlying points in a particular data set, and explain whether they are due to noise or other impurities being present in the data or due to trivial reasons. It is usually applied with the database segmentation, and is the source of true discovery, since the outliers express deviation from some previously known expectation and norm. By calculating the values of measures of current data and comparing them with previous data as well as with the normative data, the deviations can be obtained. They can be applied in forecasting, fraud detection, customer retention, etc.

Neural Networks

Neural networks are a new paradigm in computing, which involves developing mathematical structures with the ability to learn. The methods are the result of academic attempts to model the nervous system learning. Neural networks have the remarkable ability to derive meaning from complicated or imprecise data and can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. Neural networks have broad applicability to real world business problems and have already been successfully applied in many industries. Since neural networks are best at identifying patterns or trends in data, they are well suited for prediction or forecasting needs.

Neural networks use a set of processing elements (or nodes) analogous to neurons in the brain. These processing elements are interconnected in a network that can then identify patterns in data once it is exposed to the data, i.e., the network learns from experience just as people do. This distinguishes neural networks from traditional computing programs that simply follow instructions in a fixed sequential order.

Neural networks have been used successfully for classification but suffer somewhat, in that the resulting network is viewed as a black box and no explanation of the results is given. This lack of explanation inhibits confidence, the acceptance and application of results. Another problem is that neural networks suffer from long learning times, which become worse as the volume of data grows.

Genetic Algorithms

Genetic algorithms are a relatively new eomputing paradigm, inspired by Darwin's theory of evolution. A population of individuals, each representing a possible solution to a problem, is initially created at random. Then pairs of individuals combine (crossover) to produce offspring for the next generation. A mutation process is also used to randomly modify the genetic structure of some members of each new generation. The algorithm runs to generate solutions for successive generations. The probability of an individual reproducing is proportional to the goodness of the solution it represents. Hence, the quality of the solutions in successive generations improves. The process is terminated when an acceptable or optimum solution is found, or after some fixed time limit. Genetic algorithms are appropriate for problems which require optimization, with respect to some computable criterion.

This paradigm can also be applied to data mining problems. Here, the quantity to be minimized is often the number of classification errors on a training set. However, the mining of large data sets by genetic algorithms has only recently become practical due to the availability of affordable, high-speed computers. But there is hardly any special genetic algorithm designed to suit data mining problems.

Rough Sets Techniques

The rough sets theory has recently become a popular theory in the field of data mining. The theory, introduced by Pawlak in the early 1980s, provides a formal framework for the automated transformation of data into knowledge. The rough sets theory, though mathematically simple, has displayed its fruitfulness in a variety of data mining areas. A rough set is a pair of approximations—lower approximation and upper approximation sets. The lower approximation is also said to be positive cases and the upper approximation is said to be possible cases. Using these approximations, the rough set theory develops tools to discover rules from the given databases. A wide range of applications utilize the ideas of the theory. Medical data analysis, aircraft pilot performance evaluation, image processing, and voice recognition are a few examples. Almost inevitably, the database used for data mining will contain imperfections, such as noise, unknown values or errors due to inaccurate measuring equipment. The rough set theory comes handy for dealing with these types of problems, as it is a tool for handling vagueness and uncertainty inherent to decision situations. An important result from the theory is that it simplifies the search for dominating attributes leading to specific properties, or just rules pending in the data.

Support Vector Machines

Support Vector Machines (SVM) is based on statistical learning theory and is increasingly becoming useful in data mining. The main idea is to non-linearly map the data set into a high-dimensional feature space and use a linear discriminator to classify the data. Its success has been demonstrated in the areas of regression, classification and decision-tree construction. Unlike other classification techniques, which attempt to minimize the error of classification, SVMs incorporate structured risk minimization which minimizes an upper bound on the generalized error. Consider a simple case when two sets A and B are linearly separable. The idea is to determine from an infinite number of planes correctly separating A and B, the one which will have the smallest generalization error. SVMs select the plane which maximizes the margin separating the two classes. The margin is defined as the distance between the separating hyperplane to the nearest point of A, plus the distance from the hyperplane to the nearest point in B.

3.8 Other Mining Problems

We observed that a data mining system can either be a portion of a data warehousing system or a stand-alone system. Data for data mining need not always be enterprise-related data residing on a relational database. Data sources are very diverse and appear in varied form. It can be textual data, image data, CAD data, Map data, ECG data or

the much talked about Genome data. Some data are structured and some are unstructured. Data mining remains an important tool, irrespective of the forms or sources of data. We shall study the DM problems for different types of data.

Sequence Mining

Sequence mining is concerned with mining sequence data. It may be noted that in the discovery of association rules, we are interested in finding associations between items irrespective of their order of occurrence. For example, we may be interested in the association between the purchase of a particular brand of soft drinks and the occurrence of stomach upsets. But it is more relevant to identify whether there is some pattern in the stomach upsets which occurs after the purchase of the soft drink. Then one is inclined to infer that the soft drink causes stomach upsets. On the other hand, if it is more likely that the purchase of the soft drink follows the occurrence of the stomach upset, then it is probable that the soft drink provides some sort of relief to the user. Thus, the discovery of temporal sequences of events concerns causal relationships among the events in a sequence. Another application of this domain concerns drug misuse. Drug misuse can occur unwittingly, when a patient is prescribed two or more interacting drugs within a given time period of each other. Drugs that interact undesirably are recorded along with the time frame as a pattern that can be located within the patient records. The rules that describe such instances of drug misuse are then successfully inducted based on medical records.

Another related area which falls into the larger domain of temporal data mining is *trend discovery*. One characteristic of sequence-pattern discovery in comparison with trend discovery is the lack of shapes, since the causal impact of a series of events cannot be shaped.

Web Mining

With the huge amount of information available online, the World Wide Web is a fertile area for data mining research. Web mining research is at the crossroads of research from several research communities, such as database, information retrieval, and within AI, especially the subareas of machine learning and natural language processing. Web mining is the use of data mining techniques to automatically discover and extract information from web documents and services. This area of research is so huge today partly due to the interests of various research communities, the tremendous growth of information sources available on the web and the recent interest in e-commerce. This phenomenon often creates confusion when we ask what constitutes web mining. Web mining can be broken down into following subtasks:

1. Resource finding: retrieving documents intended for the web.

2. Information selection and preprocessing: automatically selecting and preprocessing specific information from resources retrieved from the web.
3. Generalization: to automatically discover general patterns at individual web sites as well as across multiple sites.
4. Analysis: validation and/or interpretation of the mined patterns.

Text Mining

The term text mining or KDT (Knowledge Discovery in Text) was first proposed by Feldman and Dagan in 1996. They suggest that text documents be structured by means of information extraction, text categorization, or applying NLP techniques as a preprocessing step before performing any kind of KDTs. Presently the term text mining, is being used to cover many applications such as text categorization, exploratory data analysis, text clustering, finding patterns in text databases, finding sequential patterns in texts, IE (Information Extraction), empirical computational linguistic tasks, and association discovery.

Spatial Data Mining

Spatial data mining is the branch of data mining that deals with spatial (location) data. The immense explosion in geographically-referenced data occasioned by developments in IT, digital mapping, remote sensing, and the global diffusion of GIS, places demands on developing data driven inductive approaches to spatial analysis and modelling. Spatial data mining is regarded as a special type of data mining that seeks to perform similar generic functions as conventional data mining tools, but modified to take into account the special features of spatial information.

For example, we may wish to discover some association among patterns of residential colonies and topographical features. A typical spatial association may look like: 'The residential land pockets are dense in a plain region and rocky areas are thinly populated"; or, "The economically affluent citizens reside in hilly, secluded areas whereas the middle income group residents prefer having their houses near the market".

3.9 Issues and Challenges in DM

Data mining systems depend on databases to supply the raw input and this raises problems, such as that databases tend to be dynamic, incomplete, noisy and large. Other problems arise as a result of the inadequacy and irrelevance of the information stored. The difficulties in data mining can be categorized as

- Limited information
- Noise or missing data

- User interaction and prior knowledge
- Uncertainty
- Size, updates and irrelevant fields

Limited information A database is often designed for purposes other then that of data mining and, sometimes, some attributes which are essential for knowledge discovery of the application domain are not present in the data. Thus, it may be very difficult to discover significant knowledge about a given domain.

Noise and missing data Attributes that rely on subjective or measurement judgments can give rise to errors, such that some examples may be misclassified. Missing data can be treated in a number of ways—simply disregarding missing values, omitting corresponding records, inferring missing values from known values, and treating missing data as a special value to be included additionally in the attribute domain. The data should be cleaned so that it is free of errors and missing data.

User interaction and prior knowledge An analyst is usually not a KDD expert, but simply a person making use of the data by means of the available KDD techniques. Since the KDD process is by definition interactive and iterative, it is challenging to provide a high performance, rapid-response environment that also assists the users in the proper selection and matching of the appropriate techniques to achieve their goals. There needs to be more human–computer interaction and less emphasis on total automation, which supports both the novice and expert users. The use of domain knowledge is important in all steps of the KDD process. It would be convenient to design a KDD tool which is both interactive and iterative.

Uncertainty This refers to the severity of error and the degree of noise in the data. Data precision is an important consideration in a discovery system.

Size, updates and irrelevant fields Databases tend to be large and dynamic, in that their contents are keep changing as information is added, modified or removed. The problem with this, from the perspective of data mining, is how to ensure that the rules are up-to-date and consistent with the most current information.

3.10 DM Application Areas

The discipline of data mining is driven in part by new applications which require new capabilities that are not currently being supplied by today's technology. These new applications can be naturally divided into three broad categories [Grossman, 1999].

A. Business and E-commerce Data

This is a major source category of data for data mining applications. Back-office, front-office, and network applications produce large amounts of data about business processes. Using this data for effective decision making remains a fundamental challenge.

Business Transactions

Modern business processes are consolidating with millions of customers and billions of their transactions. Business enterprises require necessary information for their effective functioning in today's competitive world. For example, they would like know: "Is this transaction fraudulent?"; "Which customer is likely to migrate?", and "What product is this customer most likely to buy next?'.

Electronic Commerce

Not only does electronic commerce produce large data sets in which the analysis of marketing patterns and risk patterns is critical but, it is also important to do this in near-real time, in order to meet the demands of online transactions.

B. Scientific, Engineering and Health Care Data

Scientific data and metadata tend to be more complex in structure than business data. In addition, scientists and engineers are making increasing use of simulation and systems with application domain knowledge.

Genomic Data

Genomic sequencing and mapping efforts have produced a number of databases which are accessible on the web. In addition, there are also a wide variety of other online databases. Finding relationships between these data sources is another fundamental challenge for data mining.

Sensor Data

Remote sensing data is another source of voluminous data. Remote sensing satellites and a variety of other sensors produce large amounts of geo-referenced data. A fundamental challenge is to understand the relationships, including causal relationships, amongst this data.

Simulation Data

Simulation is now accepted as an important mode of science, supplementing theory and experiment. Today, not only do experiments produce huge data sets, but so do simulations. Data mining and, more generally, data intensive computing is proving to be a critical link between theory, simulation, and experiment.

Health Care Data

Hospitals, health care organizations, insurance companies, and the concerned government agencies accumulate large collections of data about patients and health care-related details. Understanding relationships in this data is critical for a wide variety of problems—ranging from determining what procedures and clinical protocols are most effective, to how best deliver health care to the maximum number of people.

Web Data

The data on the web is growing not only in volume but also in complexity. Web data now includes not only text, audio and video material, but also streaming data and numerical data.

Multimedia Documents

Today's technology for retrieving multimedia items on the web is far from satisfactory. On the other hand, an increasingly large number of matters are on the web and the number of users is also growing explosively. It is becoming harder to extract meaningful information from the archives of multimedia data as the volume grows.

Data Web

Today, the web is primarily oriented toward documents and their multimedia extensions. HTML has proved itself to be a simple, yet powerful, language for supporting this. Tomorrow, the potential exists for the web to prove equally important for working with data. The Extensible Markup Language (XML) is an emerging language for working with data in networked environments. As this infrastructure grows, data mining is expected to be a critical enabling technology for the emerging data web.

3.11 DM Applications—Case Studies

There is a wide range of well-established business applications for data mining. These include customer attrition, profiling, promotion forecasting, product cross-selling, fraud detection, targeted marketing, propensity analysis, credit scoring, risk analysis, etc. We shall now discuss a few mock case-studies and areas of DM applications.

Housing Loan Prepayment Prediction

A home-finance loan actually has an average life-span of only 7 to 10 years, due to prepayment. Prepayment means that the loan is paid off early, rather than at the end of, say, 25 years. People prepay loans when they refinance or when they sell their home. The financial return that a home-finance institution derives from a loan depends on its life-span. Therefore, it is necessary for the financial institutions to be able to predict the life-spans of their loans. Rule discovery techniques can be used to accurately

predict the aggregate number of loan prepayments in a given quarter (or, in a year), as a function of prevailing interest rates, borrower characteristics, and account data. This information can be used to finetune loan parameters such as interest rates, points, and fees, in order to maximize profits.

Mortgage Loan Delinquency Prediction

Loan defaults usually entail expenses and losses for the banks and other lending institutions. Data mining techniques can be used to predict whether or not a loan would go delinquent within the succeeding 12 months, based on historical data, on account information, borrower demographics, and economic indicators. The rules can be used to estimate and finetune loan loss reserves and to gain some business insight into the characteristics and circumstances of delinquent loans. This will also help in deciding the funds that should be kept aside to handle bad loans.

Crime Detection

Crime detection is another area one might immediately associate with data mining. Let us consider a specific case: to find patterns in 'bogus official' burglaries.

A typical example of this kind of crime is when someone turns up at the door pretending to be from the water board, electricity board, telephone department or gas company. Whilst they distract the householder, their partners will search the premises and steal cash and items of value. Victims of this sort of crime tend to be the elderly. These cases have no obvious leads, and data mining techniques may help in providing some unexpected connections to known perpetrators.

In order to apply data mining techniques, let us assume that each case is filed electronically, and contains descriptive information about the thieves. It also contains a description of their *modus operandi*. We can use any of the clustering techniques to examine a situation where a group of similar physical descriptions coincide with a group of similar *modus operandi*. If there is a good match here, and the perpetrators are known for one or more of the offences, then each of the unsolved cases could have well been committed by the same people.

By matching unsolved cases with known perpetrators, it would be possible to clear up old cases and determine patterns of behaviour. Alternatively, if the criminal is unknown but a large cluster of cases seem to point to the same offenders, then these frequent offenders can be subjected to careful examination.

Store-level Fruits Purchasing Prediction

A super market chain called 'FruitWorld' sells fruits of different types and it purchases these fruits from the wholesale suppliers on a day-to-day basis. The problem is to

analyze fruit-buying patterns, using large volumes of data captured at the 'basket' level. Because fruits have a short shelf-life, it is important that accurate store-level purchasing predictions should be made to ensure optimum freshness and availability. The situation is inherently complicated by the 'domino' effect. For example, when one variety of mangoes is sold out, then sales are transferred to another variety. With the help of data mining techniques, a thorough understanding of purchasing trends enables a better availability of fruits and greater customer satisfaction.

Other application Areas

Risk Analysis

Given a set of current customers and an assessment of their risk-worthiness, develop descriptions for various classes. Use these descriptions to classify a new customer into one of the risk categories.

Targeted Marketing

Given a database of potential customers and how they have responded to a solicitation, develop a model of customers most likely to respond positively, and use the model for more focussed new customer solicitation. Other applications are to identify buying patterns from customers; to find associations among customer demographic characteristics, and to predict the response to mailing campaigns

Customer Retention

Given a database of past customers and their behaviour prior to attrition, develop a model of customers most likely to leave. Use the model for determining the best course of action for these customers.

Portfolio Management

Given a particular financial asset, predict the return on investment to determine the inclusion of the asset in a folio or not.

Brand Loyalty

Given a customer and the product he/she uses, predict whether the customer will switch brands.

Banking

The application areas in banking are:

- detecting patterns of fraudulent credit card use
- identifying 'loyal' customers
- predicting customers likely to change their credit card affiliation
- determine credit card spending by customer groups

- finding hidden correlations between different financial indicators
- identifying stock trading rules from historical market data

3.12 Conclusion

Successful applications of data mining are increasingly appearing in diverse applications. These are driven mainly by a glut in databases that have grown to surpass raw human processing ability. Some success stories in data mining applications in industries are reported by Brachman *et al.* [Brachman, 1996], and in scientific analysis by Fayyad *et al.* [Fayyad, 1996]. Driving the growth of this field are strong forces that are a product of the data overload phenomenon. In recent years, government agencies and industrial enterprises are using the web as the medium of publication. As a result, a very large collection of documents, images and other forms of data in structured, semistructured and unstructured forms are available on the web. Demands are being posed by the web-user community to provide data mining solutions to explore the massive repository of hyperlink data. On the other side, the database community have evolved themselves to address database-related solutions, not only for data in a DBMS but also for data in the form of time-series, sequence, temporal and spatial data. These developments will not only ensure the emergence of a new engineering discipline, but will also provide a real-world test platform for any theoretical research that results from such studies.

One, of course, cannot ignore the fact that the fundamental problems of data analysis are still as difficult as they were in the past. In the following chapters, we shall elaborate upon some of the major data mining techniques.

Current Trends Affecting DM

Five external trends are identified by Grossman [Grossman, 1999], which promise to have a fundamental impact on data mining. We shall briefly discuss these trends.

Data Trends

Perhaps the most fundamental external trend is the explosion of digital data over the past two decades. The amount of data has probably grown between six to ten orders of magnitude. Much of this data is now available on the web or accessible through networks. In order to avoid dumping such data in archive files without any practical use, techniques such as data mining must be developed, which can automate this data, and extract meaningful knowledge.

Hardware Trends

Data mining requires numerically and statistically intensive computations on large data sets. The increasing memory and processing speed of workstations enables the mining of data sets using current algorithms and techniques that were too large to be mined just a few years ago.

Network Trends

With the next generation internet connectivity, it becomes possible to correlate distributed datasets using current algorithms and techniques. In addition, new protocols, algorithms, and languages can facilitate distributed data mining using current and next generation networks.

Scientific Computing Trends

Data mining and knowledge discovery play an important role in linking theory, experiment and simulation, especially for those cases in which the experiment or simulation results in large datasets.

Business Trends

Today's businesses must be more profitable, react quicker, and offer higher quality services than ever before, and do it all using fewer people and at a lower cost. With these types of expectations and constraints, data mining becomes a fundamental technology in enabling businesses to predict opportunities and risks generated by their customers and their customers' transactions more accurately.

Further Reading

The detailed bibliography for most of the major techniques are given at the end of each chapter. This chapter deals with the general concept of data mining.

The Bayesian Network approach to data mining is introduced in [Heckerman, 1997]. P-. Shapiro and Frawley give some definition of data mining. Grossman *et al.* [Grossman, 1999] and Virmani [Virmani, 1998] identify the trends in data mining. The special issue in *Communication ACM* in 1996 [Fayyad, 1996] provides an interesting reading of the general subject of data mining and its applications. Nestorov outlines the different types of integrating RDBMS with data mining. IBM group also have investigated the possible integration of data mining with DB2.

EXERCISES

1. What is data mining?
2. How is data mining different from KDD?
3. Discuss the role of data mining in data warehousing.
4. Discuss different data mining tasks.
5. Give a brief account of data mining techniques.
6. What is spatial data mining?
7. What is sequence mining?
8. What is web mining?
9. What is text mining?
10. Give some examples where temporal mining is relevant.
11. Discuss the applications of data mining in the banking industry.
12. Discuss the applications of data mining in customer relationship management.
13. What is clustering of data? Distinguish supervised and unsupervised classifications.
14. What do you understand by deviation detection? Is it similar to association rule discovery? Give a justification.
15. Write an essay on 'Data Mining: Concepts, Issues and Trends".
16. "Text mining is different from conventional data mining." Comment.
17. What are the different application areas of data mining?
18. How can you link data mining with a DBMS.
19. How is data mining relevant to scientific data?
20. How is data mining relevant for web-based computing?
21. Discuss the usefulness of data mining in e-commerce.
22. Can we do data mining on the data generated by the web? Justify.
23. What are different ways of interfacing a data mining system with database systems?
24. Discuss the possibilities of formulating data mining tasks using SQL-92.
25. Discuss the application of data mining in science data.
26. How is sequence mining different from mining of data in DBMS?
27. What are 'nuggets' of information? Why are they important?

BIBLIOGRAPHY

Agarwal S., Agarwal R., Deshpande P.M., and Gupta A. On the computation of multidimensional aggregates. *VLDB*, 1998.

Agrawal R, Gupta A., and Sarawagi S. Modelling multidimensional databases. *ICDE*, 1997.

baskets. The discovery of such association rules can help the retailer develop marketing strategies, by gaining insight into matters like "which items are most frequently purchased by customers". It also helps in inventory management, sale promotion strategies, etc.

It is widely accepted (except for some recent developments) that the discovery of association rules is solely dependent on the discovery of frequent sets. Thus, a majority of the algorithms are concerned with efficiently determining the set of frequent itemsets in a given set of transactions database. The problem is essentially to compute the frequency of occurrences of each itemset in the database. Since the total number of itemsets is exponential in terms of the number of items, it is not possible to count the frequencies of these sets by reading the database in just in one pass. The number of counters are too many to be maintained in a single pass. As a result, having multiple passes for generating all the frequent itemsets is unavoidable. Thus, different algorithms for the discovery of association rules aim at reducing the number of passes by generating candidate sets, which are likely to be frequent sets. In other words, these algorithms attempt to eliminate, as early as possible, all those sets which can be estimated to be infrequent sets. The algorithms differ from one another in the method of handling the candidate sets and the method of reducing the number of database passes. However, one of the very recent algorithms attempts to discover frequent sets without having to generate candidate sets.

There are other related problems in this context. One can ask whether it is possible to find association rules incrementally. The algorithms for discovering frequent sets are not directly suitable when the underlying database is incremented intermittently. The idea is to avoid computing the frequent sets afresh for the incremented set of data. The concept of border sets becomes very important in this context. The sets of border sets can be determined without any extra efforts. And if no border set is affected by a database update, then it is shown that the set of frequent sets can be found without having search through the whole database.

In real-life applications, the number of frequent sets are very large in number and, as a result, the number of association rules are also too large to be useful. It then becomes meaningful to generate only those association rules in which the user is interested. The user can specify the constraint to indicate which combinations of items are the subject of his interest. The discovery of frequent itemsets with item constraints is one such important related problem, which is discussed in this chapter.

In this chapter, we shall study the important methods of discovering association rules in a large database. The main aim of this chapter is to introduce different data mining tasks related to the association rule discovery and to study different approaches to discover such rules. In Section 4.1, we shall introduce the basic concepts of frequent sets, border sets, etc., and in Section 4.4, the most popular algorithm for the discovery of frequent sets, the *A priori* Algorithm, is discussed. Section 4.5 outlines the Partition Algorithm and in Section 4.6, we present the Pincer-Search Algorithm. The Dynamic

Itemset Counting Algorithm is given in the next section. The most recent one, the FP-Tree Growth Algorithm is discussed in Section 4.8. In the following sections, we shall study the other related problems, such as incremental computation of following frequent sets and frequent sets with item constraints.

4.2 What is an Association Rule?

In this section, the basic definitions and concepts of the association rule are introduced.

Let $A = \{l_1, l_2, \ldots l_m\}$ be a set of items. Let T, the transaction database, be a set of transactions, where each transaction t is a set of items. Thus, t is a subset of A.

Definition 4.1 Support

A transaction t is said to *support* an item l_i, if l_i is present in t. t is said to support a subset of items $X \subseteq A$, if t supports each item l in X. An itemset $X \subseteq A$ has a **support** s in T, denoted by $s(X)_T$, if s% of transactions in T support X.

The support can also be defined as a fractional support, which means the proportion of transactions supporting X in T. It can also be defined in terms of absolute number of transactions supporting X in T. In the present text, unless specified otherwise, we assume the support to be %-support. We shall often drop the subscript T in the expression $s(X)_T$, when T is obviously implied. For absolute support, we refer it to as *support count*

Example 4.1

Let us consider the following set of transactions in a bookshop.

We shall look at a set of only 6 transactions of purchases of books. In the first transaction, purchases are made of books on Compiler Construction, Databases, Theory of Computations, Computer Graphics and Neural Networks; we shall denote these subjects by CC, D, TC, CG and ANN, respectively. Thus, we describe the 6 transactions as follows;

t_1 := { ANN, CC, TC, CG}

t_2 := {CC, D, CG}

t_3 := {ANN, CC, TC, CG}

t_4 := {ANN, CC, D, CG}

t_5 := {ANN, CC, D, TC, CG}

t_6 := {CC, D, TC}.

So $A := \{ANN, CC, D, TC, CG\}$ and $T := \{t_1, t_2, t_3, t_4, t_5, t_6\}$.

We can see that t_2 supports the items CC, D and CG. The item D is supported by 4 out of 6 transactions in T. Thus, the support of D is 66.6%.

DEFINITION 4.2 ASSOCIATION RULE

For a given transaction database T, an **association rule** is an expression of the form $X \Rightarrow Y$, where X and Y are subsets of A and $X \Rightarrow Y$ holds with ***confidence*** τ, if τ% of transactions in D that support X also support Y. The rule $X \Rightarrow Y$ has ***support*** σ in the transaction set T if σ% of transactions in T support $X \cup Y$.

The intuitive meaning of such a rule is that a transaction of the database which contains X tends to contain Y. Given a set of transactions, T, the problem of mining association rules is to discover all rules that have support and confidence greater than or equal to the user-specified minimum support and minimum confidence, respectively.

EXAMPLE 4.1 (CONTD.)

Consider the example of the bookshop. Assume that $\sigma = 50\%$ and $\tau = 60\%$. Clearly, ANN $\Rightarrow$ CC holds. The confidence of this rule is, in fact, 100%, because all the transactions that support ANN also support CC. On the other hand, CC $\Rightarrow$ ANN also holds but its confidence is 66%.

There is a famous myth in data mining that involves an apocryphal discovery that the sales of beer and diapers are correlated. Although it probably never happened, we will use *diapers* and *beer* as example in our discussion. Association (and sequencing) tools discover rules like

When people buy diapers they also buy beer 50% of the time.

Each rule has a *left-hand side* (buy diapers) and a *right-hand side* (buy beer). The left-hand side is also called the antecedent and the right-hand side is also called the consequent. In general, both the left-hand side and the right-hand side can contain multiple items, but for simplicity we will use single items for now.

As we have defined above, an association rule has two measures, called *confidence* and *support*[1]. To see what these terms mean and how they are computed, consider the following example of beer and diapers.

Let T consist of 500,000 transactions; 20,000 transactions of these contain diapers; 30,000 transactions contain beer; 10,000 transactions contain both diapers and beer.

Support (or prevalence) measures how often beer and diapers occur together as a percentage of the total transactions, 2% in this case (10,000/500,000). Confidence (or

[1] Some, Silicon Graphics with *MineSet* for example, use the terms *predictability* and *prevalence* instead of confidence and support.

predictability) measures how much a particular item is dependent on another. Since 20,000 transactions contain diapers and 10,000 also contain beer, when people buy diapers, they also buy beer ½ or 50% of the time.

The *inverse* rule has a confidence of 33.333% (computed as 10,000/30,000) and would be stated as

When people buy beer they also buy diapers 1/3 of the time.

Note that these two rules have the same support. Support does not depend on the direction (or implication) of the rule; it is only dependent on the set of items in the rule. For this reason, association tools also identify frequently occurring itemsets, whether or not any rules are being generated.

In the absence of any knowledge about what else was bought, we can also make the following assertions:

People buy diapers 4% of the time.
People buy beer 6% of the time.

These numbers (4% and 6%) are called the *expected confidence* of buying diapers or beer, respectively.

Observe that our transaction database is a very simplistic one, in the sense that we assume that an item is either present or absent in a transaction. For instance, in the above example, we do not distinguish between t_1 and t_3 even when t_1 may involve 2 books of Neural Networks, 3 books of Compiler Construction, a book of Theory of Computation and 5 books of Computer Graphics; and t_3 may have one book of each of these subjects. Theoretically, it is possible to translate any of the real-life transaction databases to this model. But it will be a cumbersome process. The first part of our study deals only with a binary database, that is, transaction records with only the presence or absence of an item. After understanding the basic concepts and techniques of this simple model, we shall study different generalizations of this mode.

4.3 Methods to Discover Association Rules

The discovery of association rules is the most well-studied problem in data mining. There are many interesting algorithms proposed recently and we shall discuss some of the important ones. One of the key features of all algorithms is that each of these methods assume that the underlying database size is enormous and they require multiple passes over the database. For disk-resident databases, this requires reading the database completely for each pass, resulting in a large number of disk-reads. In these algorithms, the effort spent in performing just the I/O may be considerable for large databases. For example, a 1 GB database will require 125,000 block reads for a single pass (for a block size of 8KB). If the algorithm requires 10 passes, this results in

a 1,250,000-block read. Assuming an average read time of 12 ms per page, the time spent in just performing the I/O is 1,250,000 × 12 ms = 4 hours.

Apart from poor response times, this problem places a huge burden on the I/O system. Usually, the data is collected by an online transaction processing system running hundreds or thousands of transactions per second. Running this algorithm under such workloads will adversely affect the transaction response time, and may even disrupt the daily database server. Over a network such as LAN, it will create network congestion problems and lead to poor resource utilization. Thus, the desirable features of any efficient algorithm are: (a) to reduce the I/O operations, and (b) at the same time be efficient in computing.

Problem Decomposition

The problem of mining association rules can be decomposed into two subproblems:

- Find all sets of items (*itemsets*) whose support is greater than the user-specified minimum support, σ. Such itemsets are called *frequent* itemsets.
- Use the frequent itemsets to generate the desired rules. The general idea is that if, say $ABCD$ and AB are frequent itemsets, then we can determine if the rule $AB \Rightarrow CD$ holds by checking the following inequality

$$\frac{s(\{A,B,C,D\})}{s(\{A,B\})} \geq \tau,$$

where $s(X)$ is the support of X in T.

Much research has been focused on the first subproblem, as the database is accessed in this part of the computation, and several algorithms have been proposed. We shall describe five important algorithms.

Definition 4.3 Frequent Set

Let T be the transaction database and σ be the user-specified minimum support. An itemset $X \subseteq$ A is said to be a *frequent itemset* in T with respect to σ, if

$$s(X)_T \geq \sigma$$

In Example 4.1, if we assume σ = 50%, then {ANN, CC, TC} is a frequent set as it is supported by at least 3 out of 6 transactions. You can see that any subset of this set is also a frequent set. On the other hand, {ANN, CC, D} is not a frequent itemset and hence, no set which properly contains this set is a frequent set.

You can also see that the set of frequent sets for a given T, with respect to a given σ, exhibits some interesting properties.

- **Downward Closure Property** Any subset of a frequent set is a frequent set.

- **Upward Closure Property** Any superset of an infrequent set is an infrequent set.

Discovering all frequent itemsets and their supports is a non-trivial problem if the cardinality of A, the set of items and the database T are large. For example, if $|A| = m$, the number of possible distinct itemsets is 2^m. The problem is to identify which of these are frequent in the given set of transactions. One way to achieve this is to set up 2^m counters, one for each distinct itemset and count the support for every itemset by scanning the database once. However, for many applications m can be more than 1,000; clearly, this approach is impractical. It should be noted that a large number of itemsets would have minimum support. Hence, it is not necessary to test the support for every itemset. Even if it is practically feasible, testing support for every possible itemset results in much wasted effort. On the other hand, we have just one counter and make a database pass to count the support for each database. This is not practical either, as we may have to carry out a very large number of I/O on the database.

To reduce the combinatorial search space, all algorithms exploit the two properties outlined above. All existing algorithms for discovering frequent sets are variants of this approach.

DEFINITION 4.4 MAXIMAL FREQUENT SET

A frequent set is a *maximal frequent* set if it is a frequent set and no superset of this is a frequent set.

DEFINITION 4.5 BORDER SET

An itemset is a *border set* if it is not a frequent set, but all its proper subsets are frequent sets.

One can see that if X is an infrequent itemset[2], then it must have a subset (not necessarily a proper subset) that is a border set. It is easy to derive a proof for this. Since X is not frequent, it is possible that it is a border set. In that case, the proof is done. Let us assume that X is not a border set too. Hence, there exists at least one proper subset of cardinality $|X|-1$ that is not frequent, say X'. If X' is a border set, then the proof is complete. Let us, hence, assume that X' is not a border set. We recursively construct $X, X', X'', \ldots$, and so on, having the common property that neither of these is a frequent set nor a border set and this construction process terminates when we get a set which is a border. This construction process must terminate in a finite number of steps as we are decreasing the size of the sets by 1 in every step. In most peculiar case, we may land up in a singleton itemset (the empty itemset is always considered to be a frequent set).

[2] I use this expression to mean an itemset which is not frequent

Note that if we know the set of all maximal frequent sets of a given T with respect to a σ, then we can find the set of all frequent sets without any extra scan of the database. Thus, the set of all maximal frequent sets can act as a compact representation of the set of all frequent sets. However, if we require the frequent sets together with their respective support values in T, then we have to make one more

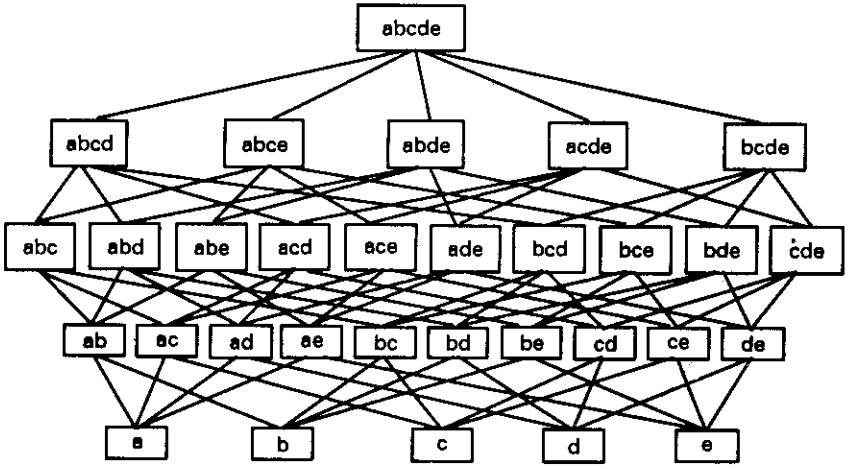


Figure 4.1 Lattice of subsets

database pass to derive the support values when the set of all maximal frequent sets is known.

We shall often refer to the lattice of subsets of A throughout this chapter. For example, if $A = \{a, b, c, d, e\}$, then the lattice is given the following figure (Figure 4.1). In this lattice, the set of maximal frequent sets acts as a boundary between the set of all frequent sets and the set of all infrequent sets. It is thus easy to characterize the class of frequent sets and the class of infrequent sets in terms of the boundary sets between these two classes. Note that some maximal frequent sets are proper subsets of some border sets. But there can also be a maximal frequent set which is not a proper subset of any border set. Similarly, it is possible that a proper subset of a border set, of cardinality one less than the border set, is not necessarily always maximal

Thus, we cannot establish a definite relationship between the set of maximal frequent sets and the set of border sets. However, the set of all border sets and the set of the maximal frequent, which are not proper subsets of any of the border sets, jointly provide a better representation of the set of frequent sets.

EXAMPLE 4.2

Study the following transaction database. We shall use this database for illustration throughout this chapter.

$A = \{A1, A2, A3, A4, A5, A6, A7, A8, A9\}$. Assume $\sigma = 20\%$. Since T contains 15 records, it means that an itemset that is supported by at least three transactions is a frequent set.

Table 4.1 Sample Database

A1	A2	A3	A4	A5	A6	A7	A8	A9
1	0	0	0	1	1	0	1	0
0	1	0	1	0	0	0	1	0
0	0	0	1	1	0	1	0	0
0	1	1	0	0	0	0	0	0
0	0	0	0	1	1	1	0	0
0	1	1	1	0	0	0	0	0
0	1	0	0	0	1	1	0	1
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	1	0
0	0	1	0	1	0	1	0	0
0	0	1	0	1	0	1	0	0
0	0	0	0	1	1	0	1	0
0	1	0	1	0	1	1	0	0
1	0	1	0	1	0	1	0	0
0	1	1	0	0	0	0	0	1

Table 4.2 Frequent Count for Some Itemsets

X	SUPPORT COUNT
{1}	2
{2}	6
{3}	6
{4}	4
{5}	8
{6}	5
{7}	7
{8}	4
{9}	2
{5, 6}	3
{5, 7}	5
{6, 7}	3
{5, 6, 7}	1

The number of transactions supporting some of the itemsets are given in Table 4.2. Please note that we are using the index i for the item A_i. We shall use this notational con vention henceforth. So {1} is not a frequent with respect to σ, but {3} is a frequent set.

It is easy to check that {5, 6, 7} is a border set; {5, 6} is a maximal frequent set; {2, 4} is also a maximal frequent set. But there is no border set having {2,4} as a proper subset. Thus, {2, 4} and {5, 6, 7} jointly represent the set of all frequent sets of T with respect to σ. This is so, because we can generate all the frequent sets from these two itemsets. If we know the set of all maximal frequent sets, we can generate all the frequent sets. Alternatively, if we know the set of border sets and the set of those maximal frequent sets, which are not subsets of any border set, then also we can generate all the frequent sets.

4.4 *A Priori* Algorithm

It is also called *the level-wise algorithm*. It was proposed by Agrawal and Srikant in 1994. It is the most popular algorithm to find all the frequent sets. It makes use of the downward closure property. As the name suggests, the algorithm is a bottom-up search, moving upward level-wise in the lattice. However, the nicety of the method is that before reading the database at every level, it graciously prunes many of the sets which are unlikely to be frequent sets.

The first pass of the algorithm simply counts item occurrences to determine the frequent itemsets. A subsequent pass, say pass k, consists of two phases. First, the frequent itemsets L_{k-1} found in the $(k-1)^{th}$ pass are used to generate the candidate itemsets C_k, using the *a priori* candidate generation procedure described below. Next, the database is scanned and the support of candidates in C_k is counted. For fast counting, we need to efficiently determine the candidates in C_k contained in a given transaction t. The set of candidate itemsets is subjected to a pruning process to ensure that all the subsets of the candidate sets are already known to be frequent itemsets. The candidate generation process and the pruning process are the most important parts of this algorithm. We shall describe these two processes separately below.

Candidate Generation

Given L_{k-1}, the set of all frequent $(k-1)$-itemsets, we want to generate a superset of the set of all frequent k-itemsets. The intuition behind the *a priori* candidate-generation procedure is that if an itemset X has minimum support, so do all subsets of X. Let us assume that the set of frequent 3-itemsets are {1, 2, 3}, {1, 2, 5}, {1, 3, 5}, {2, 3, 5}, {2, 3, 4}. Then, the 4-itemsets that are generated as candidate itemsets are the supersets of these 3-itemsets and in addition, all the 3-itemset subsets of any candidate 4-itemset (so generated) must be already known to be in L_3. The first part and part of the second part is handled by the *a priori* candidate-generation method. The following pruning algorithm prunes some candidate sets which do not meet the second criterion.

The candidate-generation method is described below.

---

gen_candidate_itemsets *with the given* L_{k-1} as follows:

$C_k = \varnothing$
for all itemsets $l_1 \in L_{k-1}$ ***do***
for all itemsets $l_2 \in L_{k-1}$ ***do***
    if $l_1[1] = l_2[1] \wedge l_1[2] = l_2[2] \wedge \ldots \wedge l_1[k-1] < l_2[k-1]$
    then $c = l_1[1], l_1[2] \ldots l_1[k-1], l_2[k-1]$
    $C_k = C_k \cup \{c\}$

Using this algorithm: $C_4 := \{\{1, 2, 3, 5\}, \{2, 3, 4, 5\}\}$ is obtained from $L_3 := \{\{1, 2, 3\}, \{1, 2, 5\}, \{1, 3, 5\}, \{2, 3, 5\}, \{2, 3, 4\}\}$. $\{1, 2, 3, 5\}$ is generated from $\{1, 2, 3\}$ and $\{1, 2, 5\}$. Similarly, $\{2, 3, 4, 5\}$ is generated from $\{2, 3, 4\}$ and $\{2, 3, 5\}$. No other pair of 3-itemsets satisfy the condition

$$l_1[1] = l_2[1] \wedge l_1[2] = l_2[2] \wedge \ldots \wedge l_1[k-1] < l_2[k-1]$$

PRUNING

The pruning step eliminates the extensions of $(k-1)$-itemsets which are not found to be frequent, from being considered for counting support. For example, from C_4, the itemset $\{2, 3, 4, 5\}$ is pruned, since all its 3-subsets are not in L_3 (clearly, $\{2, 4, 5\}$ is not in L). The pruning algorithm is described below.

---

prune(C_k)

for all $c \in C_k$
for all $(k-1)$-subsets d of c ***do***
    if $d \notin L_{k-1}$
    then $C_k = C_k \setminus \{c\}$

The *a priori* frequent itemset discovery algorithm uses these two functions (candidate generation and pruning) at every iteration. It moves upward in the lattice starting from level 1 till level k, where no candidate set remains after pruning.

A Priori **Algorithm**

Initialize: $k := 1$, C_1 = all the 1-itemsets;
read the database to count the support of C_1 to determine L_1
$L_1 :=$ {frequent 1-itemsets};
$k := 2$; *// k represents the pass number//*
while $(L_{k-1} \neq \varnothing)$ ***do***
begin
    $C_k :=$ ***gen_candidate_itemsets*** *with the given* L_{k-1}

prune(C_k)
for all transactions $t \in T$ ***do***
increment the count of all candidates in C_k that are contained in t;
L_k := All candidates in C_k with minimum support ;
$k := k + 1$;
end
Answer := $\cup_k L_k$;

A Priori Algorithm by Example

We illustrate the working of the algorithm with Example 4.2 discussed above.

k:= 1

Read the database to count the support of 1-itemsets(Table 4.3). The frequent 1-itemsets and their support counts are given below.

Table 4.3

{1}	2
{2}	6
{3}	6
{4}	4
{5}	8
{6}	5
{7}	7
{8}	4
{9}	2

L_1 := { {2}→6. {3}→6, {4}→4, {5}→8, {6}→5, {7}→7, {8}→4 }.

k:= 2

In the *candidate generation* step, we get

C_2 := {{2,3}, {2,4}, {2, 5}, {2,6}, {2, 7}, {2, 8}, {3, 4}, {3, 5}, {3, 6}, {3, 7}, {3, 8}, {4, 5}, {4, 6}, {4,7}, {4, 8},{5, 6}, {5, 7}, {5, 8}, {6,7}, {6, 8}, {7, 8}}

The *Pruning* step does not change C_2.

Read the database to count the support of elements in C_2 to get

L_2 := { {2, 3}→3, {2,4}→3, {3, 5}→3, {3, 7}→3, {5, 6}→3, {5, 7}→5, {6, 7}→3 }

k:=3

In the *candidate generation* step,

using {2,3} and {2, 4} , we get {2, 3, 4}

using {3, 5} and {3, 7}, we get {3, 5, 7} and
similarly from {5, 6} and {5, 7}, we get {5, 6, 7}.

$C_3 := \{\{2, 3, 4\}, \{3, 5, 7\}, \{5, 6, 7\}\}$.

The *Pruning* step prunes {2, 3, 4} as not all subsets of size 2, i.e., {2, 3}, {2, 4}, {3, 4} are present in L_2. The other two itemsets are retained.

Thus the pruned C_3 is {{3, 5, 7}, {5, 6, 7}}.

Read the database to count the support of the itemsets in C_3 to get

$L_3 := \{\{3, 5, 7\} \rightarrow 3\}$.

k:= 4

Since L_3 contains only one element, C_4 is empty and hence the algorithm stops, *returning* the set of frequent sets along with their respective support values as

$L := L_1 \cup L_2 \cup L_3$

4.5 Partition Algorithm

The partition algorithm is based on the observation that the frequent sets are normally very few in number compared to the set of all itemsets. As a result, if we partition the set of transactions to smaller segments such that each segment can be accommodated in the main memory, then we can compute the set of frequent sets of each of these partitions. It is assumed that these sets (set of local frequent sets) contain a reasonably small number of itemsets. Hence, we can read the whole database (the unsegmented one) once, to count the support of the set of all local frequent sets.

The partition algorithm uses two scans of the database to discover all frequent sets. In one scan, it generates a set of all potentially frequent itemsets by scanning the database once. This set is a superset of all frequent itemsets, i.e., it may contain false positives; but no false negatives are reported. During the second scan, counters for each of these itemsets are set up and their actual support is measured in one scan of the database.

The algorithm executes in two phases. In the first phase, the partition algorithm logically divides the database into a number of non-overlapping partitions. The partitions are considered one at a time and all frequent itemsets for that partition are generated. Thus, if there are n partitions, Phase I of the algorithm takes n iterations. At the end of Phase I, these frequent itemsets are merged to generate a set of all potential frequent itemsets. In this step, the local frequent itemsets of same lengths from all n partitions are combined to generate the global candidate itemsets. In Phase II, the actual support for these itemsets are generated and the frequent itemsets are identified. The

algorithm reads the entire database once during Phase I and once during Phase II. The partition sizes are chosen such that each partition can be accommodated in the main memory, so that the partitions are read only once in each phase.

A partition P of the databases refers to any subset of the transactions contained in the database. Any two partitions are non-overlapping. We define *local support* for an itemset as the fraction of the transaction containing that particular itemset in a partition. We define a local frequent itemset as an itemset whose local support in a partition is at least the user-defined minimum support. A local frequent itemset may or may not be frequent in the context of the entire database.

Partition Algorithm

P = partition_database(T); n = Number of partitions
// Phase I
for i = 1 to n ***do begin***
read_in_partition(T_i in P)
L^i = generate all frequent itemsets of T_i using *a priori* method in main memory.
end

// Merge Phase
for ($k = 2$; $L_k^i \neq \emptyset$, $i = 1, 2, ..., n$; k++) ***do begin***

$$C_k^G = \bigcup_{i=1}^{n} L_i^k$$

end

// Phase II
for i = 1 to n ***do begin***
read_in_partition(T_i in P)
for all candidates $c \in C^G$ compute $s(c)_{T_i}$
end
$L^G = \{c \in C^G \mid s(c)_{T_i} \geq \sigma\}$
Answer = L^G

The partition algorithm is based on the premise that the size of the global candidate set is considerably smaller than the set of all possible itemsets. The intuition behind this is that the size of the global candidate set is bounded by n times the size of the largest of the set of locally frequent sets. For sufficiently large partition sizes, the number of local frequent itemsets is likely to be comparable to the number of frequent itemsets generated for the entire database. If the data characteristics are uniform across partitions, then large numbers of itemsets generated for individual partitions may be common.

EXAMPLE 4.3

Let us take same database T, given in Example 4.2, and the same σ. Let us partition, for the sake of illustration, T into three partitions T_1, T_2, and T_3, each containing 5 transactions. The first partition T_1 contains transactions 1 to 5, T_2 contains transactions 6 to 10

and, similarly, T_3 contains transactions 11 to 15. We fix the local support as equal to the given support, that is 20%. Thus, $\sigma_1 = \sigma_2 = \sigma_3 = \sigma = 20\%$. Any itemset that appears in just one of the transactions in any partition is a local frequent set in that partition.

The local frequent sets of the T_1 partition are the itemsets X, such that $s(X)_{T_1} \geq \sigma_1$.

L^1 := { {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {1, 5}, {1, 6}, {1, 8}, {2, 3}, {2, 4}, {2, 8}, {4, 5}, {4, 7}, {4, 8}, {5, 6}, {5, 8}, {5, 7}, {6, 7}, {6, 8}, {1,6, 8}, {1,5,6}, {1,5,8}, {2,4,8}, {4,5,7}, {5,6,8}, {5,6,7}, {1,5,6,8} }

Similarly,

L^2 := { {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}, {2,3}, {2,4}, {2,6}, {2,7}, {2,9}, {3,4}, {3, 5}, {3, 7}, {5, 7}, {6,7}, {6,9}, {7,9}, {2,3,4}, {2,6,7}, {2,6,9}, {2,7,9}, {3, 5, 7}, {2,6,7,9} }

L^3 := { {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}, {1,3}, {1,5}, {1,7}, {2,3}, {2,4}, {2,6}, {2,7}, {2,9}, {3,5}, {3,7},{3,9},{4,6}, {4,7}, {5,6}, {5,7}, {5,8}, {6,7}, {6,8}, {1,3,5}, {1,3,7}, {1,5,7}, {2,3,9}, {2,4,6}, {2,4,7}, {3,5,7}, {4,6,7}, {5,6,8}, {1, 3, 5, 7}, {2, 4, 6, 7} }

In Phase II, we have the candidate set as

$C := L^1 \cup L^2 \cup L^3$

C := { {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}, {1,3}, {1, 5}, {1, 6}, {1,7}, {1, 8}, {2,3}, {2, 4}, {2,6}, {2,7}, {2, 8}, {2,9},{3,4}, {3,5}, {3,7}, {3,9}, {4, 5}, {4,6}, {4,7}, {4, 8}, {5, 6},{5,7}, {5, 8}, {5, 7}, {6, 7}, {6, 8}, {6,9}, {7,9},{1,3,5}, {1,3,7}, {1,5,6}, {1,5,7}, {1,5,8}, {1,6,8}, {2,3,4}, {2,3,9}, {2,4,6}, {2,4,7}, {2,4,8}, {2,6,7}, {2,6,9}, {2,7,9}, {3,5,7}, {4,5,7}, {4,6,7}, {5,6,8}, {5,6,7}, {1,5,6,8}, {2,6,7,9}, {1, 3, 5, 7}, {2, 4, 6, 7} }

Read the database once to compute the global support of the sets in C and get the final set of frequent sets.

Remarks

It is worthwhile to discuss here certain aspects of local support vs global support. We saw, in the previous example, that the local support is the same as the global support. This implies that if an itemset is not frequent in any of the segments, then it is not frequent in the whole database either.

4.6 Pincer-Search Algorithm

One can see that the *a priori* algorithm operates in a bottom-up, breadth-first search method. The computation starts from the smallest set of frequent itemsets and moves

upward till it reaches the largest frequent itemset. The number of database passes is equal to the largest size of the frequent itemset. When any one of the frequent itemsets becomes longer, the algorithm has to go through many iterations and, as a result, the performance decreases.

A natural way to overcome this difficulty is to somehow incorporate a bi-directional search, which takes advantages of both the bottom-up as well as the top-down process. The pincer-search algorithm is based on this principle. It attempts to find the frequent itemsets in a bottom-up manner but, at the same time, it maintains a list of maximal frequent itemsets. While making a database pass, it also counts the support of these candidate maximal frequent itemsets to see if any one of these is actually frequent. In that event, it can conclude that all the subsets of these frequent sets are going to be frequent and, hence, they are not verified for the support count in the next pass. If we are lucky, we may discover a very large maximal frequent itemset very early in the algorithm. If this set subsumes all the candidate sets of level k, then we need not proceed further and thus we save many database passes. Clearly, the pincer-search has an advantage over *a priori* algorithm when the largest frequent itemset is long.

In this algorithm, in each pass, in addition to counting the supports of the candidate in the bottom-up direction, it also counts the supports of the itemsets of some itemsets using a top-down approach. These are called the *Maximal Frequent Candidate Set* (MFCS). This process helps in pruning the candidate sets very early on in the algorithm. If we find a maximal frequent set in this process, then it is recorded in the MFCS.

Consider a pass k, during which itemsets of size k are to be classified. If some itemset that is an element of the MFCS, say X, of cardinality greater than k is found to be frequent in this pass, then all its subsets must be frequent. Therefore, all of its subsets of cardinality k can be pruned from the set of candidate sets considered in the bottom-up direction in the pass. They, and their supersets, will never be candidates throughout the rest of the execution, potentially improving the performance.

Similarly, when a new infrequent itemset is found in the bottom-up direction, the algorithm will use it to update the MFCS. The subsets of the MFCS should not contain this in frequent itemsets.

The MFCS initially contains a single element, the itemset of cardinality n containing all the elements of the database. If some m 1-itemsets are infrequent after the first pass (after reading the database once), the MFCS will have one element of cardinality $n–m$. Removing the m infrequent items from the initial element of the MFCS, generates this itemset. In this case, the top-down search goes down m levels in one pass. Unlike the search in the bottom-up direction which goes in one-pass, the top-down search can go down many levels in one pass. This is because we may discover a maximal frequent set very early in the algorithm.

Pincer-Search Method

$L_0 := \varnothing$; $k := 1$; $C_1 := \{\{i\} \mid i \in I\}$; $S_0 = \varnothing$;
MFCS := $\{\{1,2, \ldots, n\}\}$; MFS := $\varnothing$;
do until $C_k = \varnothing$ and $S_{k-1} = \varnothing$
read database and count supports for C_k and MFCS;
MFS := MFS $\cup$ {frequent itemsets in MFCS};
S_k := {infrequent itemsets in C_k};
call *MFCS-gen* algorithm if $S_k \neq \varnothing$;
call *MFS-pruning* procedure;
generate candidates C_{k+1} from C_k; (similar to a priori's generate & prune)
if any frequent itemset in C_k is removed in MFS-pruning procedure
call the recovery procedure to recover candidates to C_{k+1};
call *MFCS prune procedure* to prune candidates in C_{k+1};
$k := k+1$;
return MFS

MFCS-gen

for all itemsets $s \in S_k$
for all itemsets $m \in$ MFCS
if s is a subset of m
MFCS := MFCS\{m};
for all items $e \in$ itemset s
if $m\backslash\{e\}$ is not a subset of any itemset in MFCS
MFCS := MFCS $\cup \{m\backslash\{e\}\}$;
return MFCS

Recovery

for all itemsets $l \in C_k$
for all itemsets $m \in$ MFS
if the first k–1 items in l are also in m
/* suppose $m.\text{item}_j = l.\text{item}_{k-1}$ */
for i from j+1 to $|m|$
$C_{k+1} := C_{k+1} \cup \{\{l.\text{item}_1, l.\text{item}_2, \ldots, l.\text{item}_k, m.\text{item}_i\}\}$

MFS-Prune

for all itemsets c in C_k
if c is a subset of any itemset is the current MFS
delete c from C_k;

MFCS-Prune

for all itemsets c in C_{k+1}
if c is not a subset of any itemset in the current MFCS
delete c from C_{k+1};

EXAMPLE 4.4

We shall use Example 4.2 to illustrate the working of the example.

STEP 1: $L_0 := \emptyset$; $k := 1$;

$C_1 := \{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}\}$

MFCS := {1, 2, 3, 4, 5, 6, 7, 8, 9}

MFS := ∅;

PASS ONE: Database is read to count the support as follows

{1} → 2, {2} → 6, {3} → 6, {4} → 4, {5} → 8, {6} → 5, {7} → 7, {8} → 4, {9} → 2

{1, 2, 3, 4, 5, 6, 7, 8, 9} → 0.

So MFCS := {1, 2, 3, 4, 5, 6, 7, 8, 9} and MFS := ∅;

$L_1 := \{\{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}\}$

$S_1 := \{\{1\}, \{9\}\}$

At this stage we call the MFCS-gen to update MFCS.

For {1} in S_1 and for {1, 2, 3, 4, 5, 6, 7, 8, 9} in MFCS, we get the new element in MFCS as {2, 3, 4, 5, 6, 7, 8, 9}.

For {9} in S_1 and for {2, 3, 4, 5, 6, 7, 8, 9} in MFCS, we get the new element in MFCS as {2, 3, 4, 5, 6, 7, 8}.

We generate the candidate itemsets

$C_2 := \{ \{2,3\}, \{2,4\}, \{2,5\}, \{2,6\}, \{2,7\}, \{2,8\}, \{3,4\}, \{3,5\}, \{3,6\}, \{3,7\}, \{3,8\}, \{4,5\}, \{4,6\}, \{4,7\}, \{4,8\}, \{5,6\}, \{5,7\}, \{5,8\}, \{6,7\}, \{6,8\}, \{7,8\} \}$

PASS TWO: read the database to count the support of elements in C_2 and MFCS as given below:

{2,3}→3, {2,4}→3, {2,5}→0, {2,6}→2, {2,7}→2, {2,8}→1, {3,4}→1, {3,5}→3, {3,6}→0, {3,7}→3, {3,8}→0, {4,5}→1, {4,6}→1, {4,7}→2, {4,8}→1, {5,6}→3, {5,7}→5, {5,8}→2, {6,7}→3, {6,8}→2, {7,8}→0

{2, 3, 4, 5, 6, 7, 8}→0.

MFS:= ∅;

$L_2 := \{ \{2,3\}, \{2,4\}, \{3,5\}, \{3,7\}, \{5,6\}, \{5,7\}, \{6,7\} \}$

$S_2 := \{ \{2,5\}, \{2,6\}, \{2, 7\} \{2,8\}, \{3,4\}, \{3,6\}, \{3,8\}, \{4,5\}, \{4,6\}, \{4,7\}, \{4,8\}, \{5,8\}, \{6,8\}, \{7,8\} \}$

For {2,5} in S_2 and for {2, 3, 4, 5, 6, 7, 8} in MFCS, we get the new elements in MFCS as {3, 4, 5, 6, 7, 8} and {2, 3, 4, 6, 7, 8}

For {2,6} in S_2 and for {3, 4, 5, 6, 7, 8} in MFCS, since {2,6} is not contained in this element of MFCS and hence, no action.

For {2, 3, 4, 6, 7, 8} we get two new elements in MFCS in place of {2, 3, 4, 6, 7, 8} as {3, 4, 6, 7, 8} and {2, 3, 4, 7, 8}. Since {3, 4, 6, 7, 8} is already contained in an element of MFCS, it is excluded from MFCS.

So at this stage MFCS := {{ 3, 4, 5, 6, 7, 8}, {2, 3, 4, 7, 8}}.

For {2,7} in S_2, we get

MFCS := {{3, 4, 5, 6, 7, 8}, {2, 3, 4, 8}}.

For {2,8} in S_2, we get

MFCS := {{3, 4, 5, 6, 7, 8}, {2, 3, 4}}.

For {3,4} in S_2, we get

MFCS := {{3, 5, 6, 7, 8}, {4, 5, 6, 7, 8}, {2, 3}, {2, 4}}.

For {3,6} in S_2, we get

MFCS := {{3, 5, 7, 8}, {4, 5, 6, 7, 8}, {2, 3}, {2, 4}}.

For {3,8} in S_2, we get

MFCS := {{3, 5, 7}, {4, 5, 6, 7, 8}, {2, 3}, {2, 4}}.

For {4,5} in S_2, we get

MFCS := {{3, 5, 7}, {5, 6, 7, 8}, {4, 6, 7, 8}, {2, 3}, {2, 4}}.

For {4,6} in S_2, we get

MFCS := {{3, 5, 7}, {5, 6, 7, 8}, {4, 7, 8}, {2, 3}, {2, 4}}.

For {4,7} in S_2, we get

MFCS := {{3, 5, 7}, {5, 6, 7, 8}, {4, 8}, {2, 3}, {2, 4}}.

For {4,8} in S_2, we get

MFCS := {{3, 5, 7}, {5, 6, 7, 8}, {2, 3}, {2, 4}}.

For {5,8} in S_2, we get

MFCS := {{3, 5, 7}, {6, 7, 8}, {5, 6, 7}, {2, 3}, {2, 4}}.

For {6,8} in S_2, we get

MFCS := {{7, 8}, {3, 5, 7}, {5, 6, 7}, {2, 3}, {2, 4}}.

For {7,8} in S_2, we get

MFCS := {{8}, {3, 5, 7}, {5, 6, 7}, {2, 3}, {2, 4}}.

We generate the candidate sets as

C_3 := {{2, 3, 4}, {3, 5, 7}, {5, 6, 7}}

In the pruning stage the itemsets {2, 3, 4} are pruned from C_3 and hence,

C_3 := {{3, 5, 7}, {5, 6, 7}}

At this stage we make one more pass of the database to count the supports of {{3, 5, 7}, {5, 6, 7}}.

EXAMPLE 4.5

The above example illustrates the algorithm. But you can see that it takes the same number of passes as an *a priori* algorithm. The following example illustrates (Table 4.4) the advantages of the pincer-search algorithm.

Table 4.4

A1	A2	A3	A4	A5	A6	A7	A8	A9
1	0	0	0	1	1	0	1	0
0	1	0	1	0	0	0	1	0
0	0	0	1	1	0	1	0	0
0	0	1	0	0	0	0	0	0
0	0	0	0	1	1	1	0	0
0	1	1	1	0	0	0	0	0
0	1	0	0	0	1	1	0	1
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	1	0
0	0	1	1	1	0	1	0	0
0	0	1	1	1	0	1	0	0
0	0	0	0	1	1	0	1	0
0	1	1	1	0	1	1	0	0
1	0	1	1	1	0	1	0	0
0	1	1	0	0	0	0	0	1

During the first pass, the supports counted are

{1}→2, {2}→5, {3}→7, {4}→7, {5}→8, {6}→5, {7}→7, {8}→4, {9}→2

{1, 2, 3, 4, 5, 6, 7, 8, 9}→0.

We update MFCS as {2, 3, 4, 5, 6, 7, 8}
In the second pass, we count the supports as

{23}→3, {24}→3, {25}→0, {26}→2, {27}→2, {28}→1, {34}→5, {35}→3, {36}→1, {37}→4, {38}→0, {45}→4, {46}→1, {47}→5, {48}→1, {56}→3, {57}→5, {58}→2, {67}→3, {68}→2, {78}→0.

At this stage, we update the MFCS to obtain

MFCS := {{2, 3, 4}, {3, 4, 5, 7}, {5, 6, 7}, {8}}

and C_3 := {{2, 3, 4}, {3, 4, 5}, {3, 5, 7}, {4, 5, 7}, {5, 6, 7}}

We then make a database pass to read the frequency of these itemsets. The algorithm stops at this point. It can be noted that an *a priori* algorithm would have made 4 passes and the pincer-search makes only 3 passes.

4.7 Dynamic Itemset Counting Algorithm

This algorithm was proposed by Bin *et al.* in 1997. The rationale behind DIC is that it works like a train running over the data, with stops at intervals M between transactions. When the train reaches the end of the transaction file, it has made one pass over the data, and it starts all over again from the beginning for the next pass. The passengers on the train are itemsets. When an itemset is on the train, we count its occurrence in the transactions that are read. When an *a priori* algorithm is considered in this metaphor, all itemsets get on at the start of a pass and get off at the end. The 1-itemsets take the first pass, the 2-itemsets take the second pass, and so on. In DIC, there is the added flexibility of allowing itemsets to get on at any stop as long as they get off at the same stop the next time the train goes around. Therefore, the itemset has seen all the transactions in the file.

Consider for example, if we are mining 40,000 transactions and M =10,000, we will count all the 1-itemsets in the first 40,000 transactions we read. However, we will begin counting 2-itemsets after the first 10,000 transactions have been read. We will begin counting 3-itemsets after 20,000 transactions. For now, let us assume that there are no 4-itemsets we need to count. Once we get to the end of the file, we will stop counting the 1-itemsets and go back to the start of the file to count the 2- and 3-itemsets. After the first 10,000 transactions, we will finish counting the 2-itemsets and after 20,000 transaction, we will finish counting the 3-itemsets. In total, we have made 1.5 passes over the data instead of the 3 passes a level-wise algorithm would make.

Initially, we identify certain 'stops' in the database. It is assumed that we read the records sequentially as we do in other algorithms, but pause to carry out certain computations at the 'stop' points. For notational convenience, we assign numbers to each stop sequentially.

We then define four different structures:

- Dashed Box
- Dashed Circle
- Solid Box
- Solid Circle

Each of these structures maintains a list of itemsets. Itemsets in the 'dashed' category of structures have a counter and the stop number with them. The counter is to keep track of the support value of the corresponding itemset. The stop number is to keep track whether an itemset has completed one full pass over a database.

The itemsets in the 'solid' category structures are not subjected to any counting. The itemset in the solid box is the confirmed set of frequent sets. The itemsets in the solid circle are the confirmed set of infrequent sets.

The algorithm counts the support values of the itemsets in the dashed structure as it moves along from one stop point to another.

During the execution of the algorithm, at any stop point, the following events take place:

- Certain itemsets in the dashed circle move into the dashed box. These are the itemsets whose support-counts reach σ value during this iteration (reading records between two consecutive stops).
- Certain itemsets enter afresh into the system and get into the dashed circle. These are essentially the supersets of the itemsets that move from the dashed circle to the dashed box.
- The itemsets that have completed one full pass, move from the dashed structure to solid structure. That is, if the itemset is in a dashed circle while completing a full pass, it moves to the solid circle. If it is in the dashed box, then it moves into solid box after the completing a full pass.

We shall formally describe the DIC algorithm now.

DIC Algorithm

Initially,
    Solid box contains the empty itemset;
    Solid circle is empty
    Dashed box is empty;
    Dashed circle contains all 1-itemsets with the respective stop-number as 0;
    Current stop-number := 0;

do until the dashed circle is empty

read the database till the next stop point and increase the counters of the itemsets in the dashed box and in the dashed circle as we go along, record by record, to reach the next stop.

increase the current-stop-number by 1;
for each itemset in the dashed circle

if count of the itemset is greater than σ
then move the itemset to the dashed box
generate a new itemset to be put into the dashed circle
with counter value = 0 and stop number = current stop number.
else
if its stop number is equal to the current stop number
then move this itemset to solid circle.

for each itemset in the dashed box
if its stop-number is equal to the current stop number
then move this itemset to the solid box
end
return the itemsets in solidbox

EXAMPLE 4.6

The following example (Table 4.5) illustrates the working of the DIC algorithm.

Table 4.5

A1	A2	A3	A4	A5	A6	A7	A8	A9
1	0	0	0	1	1	0	1	0
0	1	0	1	0	0	0	1	0
0	0	0	1	1	0	1	0	0
0	0	1	0	0	0	0	0	0
0	0	0	0	1	1	1	0	0
0	1	1	1	0	0	0	0	0
0	1	0	0	0	1	1	0	1
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	1	0
0	0	1	0	1	0	1	0	0
0	0	1	0	1	0	1	0	0
0	0	0	0	1	1	0	1	0
0	1	0	1	0	1	1	0	0
1	0	1	0	1	0	1	0	0
0	1	1	0	0	0	0	0	1

Step 1. The empty itemset is in the solid box.

All the 1-itemsets are in the dashed circles.

The sets {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, and {9} are in the dashed circle.

All other itemsets are free, i.e., not in any of the 4 types of boxes.

Step 2. First read 5 transactions (transactions t_1–t_5).

For each transaction, increment the counts of the respective counters of the itemsets in the dashed box or dashed circle.

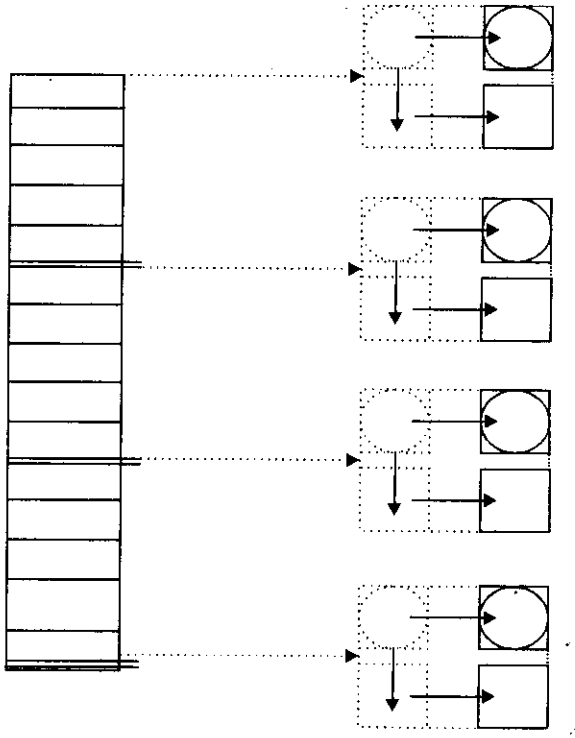


Figure 4.2 A pictorial depiction of the working of DIC. The horizontzal arrow indicates movement of the itemset after completion of database pass. The vertical arrow indicates movement of the itemset after reaching the frequency threshold. The dashed arrow indicates the new itemset entering the system at a stop-point.

The counts of the itemsets in the dashed circles are given below.

{1} = 1, {2} = 1, {3} = 1, {4} = 2, {5} = 3, {6} = 2, {7} = 2, {8} = 2, and {9} = 0.

Thus, itemset {5} is put in a dashed box and removed from the dashed circle.

Step 3. Read the next 5 transactions (transactions t_6–t_{10}).

The updated counts of the itemsets in the dashed box and in the dashed circle are

{1} = 1, {2} = 3, {3} = 3, {4} = 3, {5} = 5, {6} = 3, {7} = 4, {8} = 3, and {9} = 1.

The itemsets {2}, {3}, {4}, {5}, {6}, {7}, and {8} are now put in dashed box and are removed from the dashed circle.

The additional itemsets which are put in the dashed circle are

{2, 3}, {2, 4}, {2, 5 }, {2, 6}, {2, 7}, {2, 8}, {3, 4}, {3, 5}, {3, 6}, {3, 7}, {3, 8}, {4, 5}, {4, 6}, {4, 7}, {4, 8}, {5, 6}, {5, 7}, {5, 8}, {6, 7}, {6, 8}, {7, 8}.

We include a counter for each of these itemsets at this stage.

Step 4. Read the next 5 transactions (transactions t_{11}–t_{15}). The updated count of each of the counters are

{1} = 2, {2} = 5, {3} = 6, {4} = 4, {5} = 8, {6} = 5, {7} = 7, {8} = 4, and {9} = 2. {2, 3} = 1, {2, 4} = 1, {2, 5}= 0, {2, 6} = 1, {2, 7}= 1, {2, 8} = 0, {3, 4} = 0, {3, 5} = 2, {3, 6} = 0, {3, 7}= 2, {3, 8} = 0, {4, 5}= 0, {4, 6}= 1, {4, 7}= 1, {4, 8}= 0, {5, 6}= 1, {5, 7}= 2, {5, 8}= 1, {6, 7}= 1, {6, 8}= 1, {7, 8} = 0.

The 1-itemsets have completed their traversal of the whole set of transactions and hence, {2}, {3}, {4}, {5}, {6}, {7}, and {8} are put in the solid box and are removed from the dashed box. The itemset {1} and {9} are removed from the dashed circle and is put in the solid circle. The counters for these sets are now dropped.

Step 5. Read the first 5 transactions(t_1–t_5) and update the count of the itemsets in the dashed box and the dashed circle. The updated counts are

{2, 3} = 1, {2, 4} = 2, {2, 5}= 0, {2, 6} = 1, {2, 7}= 1, {2, 8} = 1, {3, 4} = 0, {3, 5} = 2, {3, 6} = 0, {3, 7} = 2, {3, 8} = 0, {4, 5}= 1, {4, 6}= 1,

{4, 7}= 2, {4, 8}= 1, {5, 6}= 3, {5, 7}= 4, {5, 8}= 2, {6, 7}= 2, {6, 8}= 2, {7, 8} = 0.

At this stage, {5, 6} and {5, 7} are moved from the dashed circle to the dashed box, as their count exceeds the required support.

Step 6. Read the transactions t_6–t_{10}. The updated counts are

{2, 3} = 2, {2, 4} = 3, {2, 5}= 0, {2, 6} = 2, {2, 7}= 2, {2, 8} = 1, {3, 4} = 1, {3, 5} = 3, {3, 6} = 0, {3, 7}= 3, {3, 8} = 0, {4, 5}= 0, {4, 6}= 1,

{4, 7}= 2, {4, 8}= 1, {5, 6}= 3, {5, 7}= 5, {5, 8}= 2, {6, 7}= 3, {6, 8}= 2, {7, 8} = 0.

Clearly, the itemsets {2, 4}, {3, 5}, {3, 7} and {6, 7} are moved to the solid box from the dashed circle; and {5, 6} and {5, 7} are moved from the dashed box to the solid box. The remaining itemsets (listed below) in the dashed circle are moved to the solid circle.

{2, 3}, {2, 5}, {2, 6}, {2, 7}, {2, 8}, {3, 4}, {3, 5}, {3, 6}, {3, 7}, {3, 8}, {4, 5}, {4, 6}, {4, 7}, {4, 8}, {5, 8}, {6, 8}, {7, 8}.

The counters for these sets are dropped. The itemset {5, 6, 7} is now put in the dashed circle and a counter for this is introduced.

Step 7. We need to scan the transactions t_{10}–t_{15} and the transactions t_1–t_5 to count the support of {5, 6, 7}. Thus, the algorithm requires only 2.75 database passes instead of 3 passes as in the level-wise method.

4.8 FP-TREE GROWTH ALGORITHM

Most of the above algorithms suffer from the following two shortcomings:

1. It is costly to handle large numbers of candidate sets. For instance, if there are 10^4 frequent 1-itemsets, then approximately, 10^7 candidate 2-itemsets are generated. Moreover, if there is a frequent set of size 100, then roughly 10^{30} candidate sets are generated in this process.
2. It is tedious to repeatedly scan the database and check a large set of candidates by pattern matching.

Keeping this in mind, a new class of algorithms have recently been proposed which avoids the generation of large numbers of candidate sets. We describe one such method, called the FP-Tree Growth algorithm. It is proposed by Han *et al.* The main idea of the algorithm is to maintain a Frequent Pattern Tree (FP-Tree) of the database. It is an extended prefix-tree structure, storing crucial and quantitative information about frequent sets. The tree nodes are frequent items and are arranged in a such a way that more frequently occurring nodes will have better chances of sharing nodes than the less frequently occurring ones. The method starts from frequent 1-itemsets as an initial suffix pattern and examines only its conditional pattern base (a subset of the database), which consists of the set of frequent items co-occurring with the suffix pattern. The algorithm constructs the conditional FP-tree and performs mining on this tree.

The algorithm involves two phases. In Phase I, it constructs the FP-tree with respect to a given σ. The construction of this tree requires two passes over the whole database. In Phase II, the algorithm does not use the transaction database any more, but it uses the FP-tree. Interestingly, the FP-tree contains all the information about frequent itemsets with respect to the given σ. Let us first understand the concept of the FP-tree.

DEFINITION 4.7 FP-TREE

A *frequent pattern tree* (or FP-tree) is a tree structure consisting of an item-prefix-tree and a frequent-item-header table.

- Item-prefix-tree:
  - It consists of a root node labelled null
  - Each non-root node consists of three fields:
    - Item name,
    - support count, and
    - node link.
- frequent-item-header-table: It consists of two fields;
    - Item name;
    - Head of node link which points to the first node in the FP-Tree carrying the item name.

It may be noted that the FP-tree is dependent on the support threshold σ. For different values of σ, the trees are different. There is another typical feature of the FP-tree; it depends on the ordering of the items. The ordering that is followed in the original paper is the decreasing order of the support counts. However, different ordering may offer different advantages. Thus, the header table is arranged in this order of the frequent items. Let us study the construction of a FP-tree from a transaction database.

We make one scan of the database T and compute L_1, the set of frequent 1-itemsets. For convenience, let us call this the set of frequent items. In other words, we refer to L_1 as a set of items rather than a class of singleton sets. Then we sort the items in L_1 in the decreasing order of frequency counts. From this stage onwards, the algorithm ignores all the non-frequent items from the transaction and views any transaction as a list of frequent items in the decreasing order of frequency. Without any ambiguity, we can refer to the transaction t as such a list. The first element of the list corresponding to any given transaction, is the most frequent item among the items supported by t. For a list t, we denote *head_t* as its first element and *body_t* as the remaining part of the list (the portion of the list t after removal of *head_t*). Thus, t is [*head_t* | *body_t*]. The FP-tree construction grows the tree recursively using these concepts.

FP-tree construction Algorithm

```
create a root node root of the FP-Tree and label it as null.
    do for every transaction t
        if t is not empty
        insert (t, root)
        link the new nodes to other nodes with similar labels links originating from header list.
    end do
return FP-Tree

insert(t, any_node)
    do while t is not empty
        if any_node has a child node with label head_t
```

then increment the link count between *any_node* and *head_t* by 1
else create a new child node of *any_node* with labe; *head_t* with link count 1.
call insert(*body_t*, *head_t*)
end do

EXAMPLE 4.7

Let us consider the database given in Example 4.5 (Table 4.4). The frequent items are 2, 3, 4, 5, 6, 7 and 8. If we sort them in the order of their frequency, then they appear in the order 5, 3, 4, 7, 2, 6, and 8. If the transactions are written in terms of only frequent items, then the transactions are

5, 6, 8
4, 2, 8
5, 4, 7
3
5, 7, 6
3, 4, 2
7, 2, 6
5
8
5, 3, 4, 7
5, 3, 4, 7
5, 6, 8
3, 4, 7, 2, 6
5, 3, 4, 7
3, 2.

The scan of the first transaction leads to the construction of the first branch of the tree (Figure 4.3). Notice that the branch is not ordered in the same way as the transaction appears in the database. The items are ordered according to the decreasing order of frequency of the frequent items. The complete table is given in Figure 4.4.

It is now easy to read the frequent itemsets from the FP-tree. The algorithm starts from the leaf node in a bottom-up approach. Thus, after processing item 6, it processes item 7. Let us consider, for instance, the frequent item {6}. There are four paths to 6 from the root; these are {5, 7, 6}, {5, 6}, {4, 7, 2, 6}, and {7, 2, 6}. All these paths have labels of 1. A label of a path is the smallest of the link counts of its links. Thus, each of these combinations appear just once. The paths {5, 7, 6}, {5, 6}, {4, 7, 2, 6} and {7, 2, 6} from the root to the nodes with label 6 are called the prefix subpaths of 6. The prefix subpath of a node a are the paths from the root to the nodes labelled a and the count of links along a path are adjusted by adjusting the frequency count of every link in such a path, so that they are the same as the count of the link incident on a along the

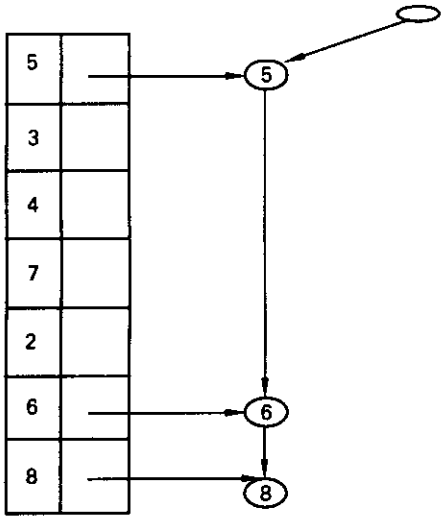


Figure 4.3 Illustration of insert (t, root) operation.

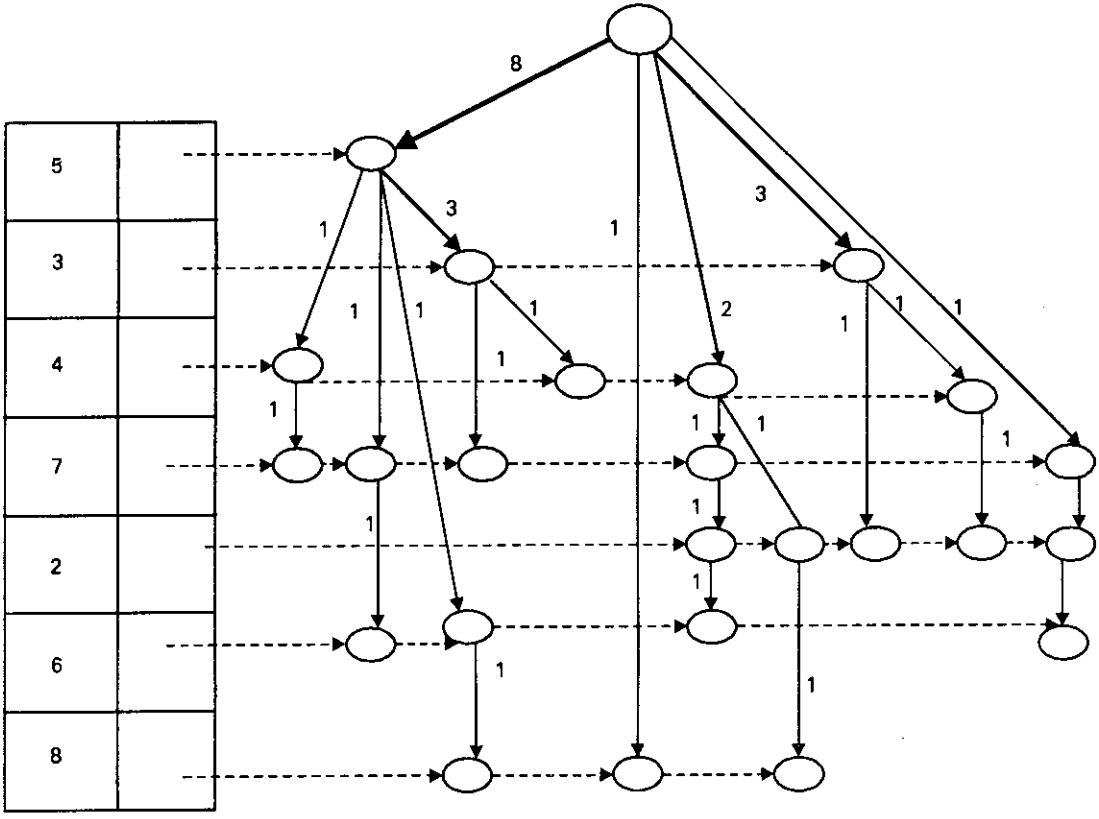


Figure 4.4 Complete FP-tree of the above example. Labels on edges represent frequency.

path. This is called a transformed prefix path. The transformed prefix paths of a node *a* form a truncated database of patterns which co-occur with *a*. This is called a conditional pattern base. Once the conditional pattern base is derived from the FP-tree, one can compute all the frequent patterns associated with it in the conditional pattern base. By creating a small conditional FP-tree for *a*, the process is recursively carried out starting from the leaf nodes.

The conditional pattern base for {6} is the following:

For the prefix subpath {5, 7, 6}, we get {5, 6}→1, {7, 6}→1, {5, 7, 6}→1.

For the prefix subpath {5, 6}, we get {5, 6}→1.

For the path {4, 7, 2, 6}, we have

{4, 6}, {7, 6}, {2, 6}, {4, 7, 6}, {4, 2, 6}, {7, 2, 6}, {4, 7, 2, 6} and for the path {7, 2, 6}, {7, 6}, {2, 6}, {7, 2, 6} all with lable 1.

Thus the only frequent pattern with prefix 6 is {7, 6}→3.

In Figure 4.5, the conditional pattern base of 7 is illustrated. Please note that since the processing now is bottom-up, the combination {6, 7} is already considered when the item 6 was considered.

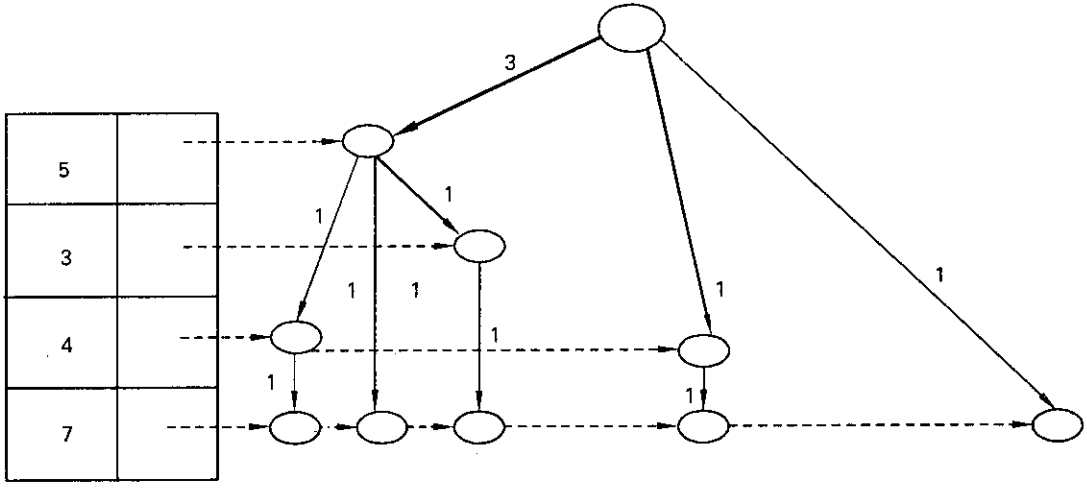


Figure 4.5 Conditional pattern base for item 7

4.9 Discussion on Different Algorithms

We observe that all the algorithms have two important and common steps: (i) candidate generation, and (ii) I/O operation. The initial candidate set generation,

especially for the frequent 2-itemsets, is the key issue to improve the performance of frequent set discovery algorithms. Another performance measure is the amount of data that has to be scanned during the discovery of the frequent itemset. The level-wise algorithm requires one pass over the database of all transactions for each iteration. The partition algorithm attempts to reduce the I/O operations by considering smaller segments of the database. The pincer-search algorithm attempts to reduce the number of candidate sets by considering Maximal Frequent Sets (MFS). But in order to get the frequency count of each frequent set, it requires one additional pass. DIC is based on a very novel idea, but in one sense it is similar in principle to the partition algorithm. The recent development of the FP-tree algorithm avoids the candidate generation steps.

There are different variations of all the algorithms that have been studied to improve the performance of mining association rules.

A hash-based technique is used to reduce the size of the candidate k-itemsets. Another variation is to reduce the number of transactions to be scanned at higher values of k. Since a transaction that does not contain any frequent k-itemset cannot contain any frequent $(k+1)$-itemset, these types of transactions can be marked during the k^{th} scanning and are not considered in the subsequent scanning. We have observed that the other major variation is to reduce the number of database passes.

Note that as k increases, not only is there a smaller number of frequent k-itemsets, but there also are fewer transactions containing any frequent k-itemsets. Reducing the number of transactions to be scanned and trimming the number of items in each transaction can improve the efficiency of the algorithms. A modification of the level-wise algorithm, *A priori*-TID attempts to handle this issue. In this algorithm, reading the complete database after the first iteration is avoided. During the first pass, the transaction id's and candidate frequent k-itemsets present in each transaction are generated in each iteration. This is used to determine the frequent sets of size $k+1$ present in each transaction during the next iteration.

Another factor that seriously affects the performance of an algorithm is the distribution of transactions over the whole database. If the data is distributed fairly uniformly throughout, then by choosing the stop points in DIC reasonably close, one can improve the performance substantially. To take care of randomness in the distribution, Toivonen proposes a sampling algorithm, where random samples of the database are taken and frequent sets with smaller support are computed for each sample. The sample size is so chosen that the sampled database can be accommodated in the main memory. Though it is an effective way of computing frequent sets, it pays the additional penalty of having to count more itemsets due to the reduced threshold.

The bi-directional search used by the pincer-search has an advantage over the bottom-up search only when there are some very large frequent itemsets. In that event, the bottom-up search may take large number of passes, whereas the pincer-search may detect a maximal frequent set very early and hence may reduce the number of passes.

Recently Shenoy *et al.* proposed an efficient algorithm – Vertical Itemset Partitioning for Efficient Rule Extraction – VIPER, which is based on vertical mining. The database can be represented either by a set of tuples, each tuple listing the presence or absence of items; or it can be represented as a list of items, each item indicating the tuples in which it is present. The latter representation is called the vertical representation, as its basic structure is a column. VIPER stores data in compressed bit-vector called 'snakes' and provides manipulations with this structure.

The next few years will witness some very innovative algorithms to improve performance. In the following sections, we shall study other problems related to the discovery of association rules.

4.10 Incremental Algorithm

In the previous sections, we have studied different algorithms for the discovery of frequent sets and for mining of association rules. We have analyzed the possibilities of reducing the I/O operations and segmenting the database into smaller fragments to increase efficiency. But one common assumption in all these approaches is that the database itself does not change. In practice, no transaction database is static. It is possible that more transactions are added intermittently to the database. Such updates to the transaction database could potentially invalidate the existing set of association rules. It is possible that the updates may also introduce more new association rules. The problem is to update the association rules when the database is updated. As we discussed earlier, the crucial step of finding association rules is to discover the set of frequent itemsets. In the present context, the problem boils down to updating the set of frequent itemsets. One simple method will be to compute afresh the set of frequent itemsets for the whole database. But in this case, all efforts have been expended in computing the frequent sets before the update is wasted. Moreover, if we can somehow make intelligent use of the earlier computation, we may be able to reduce the I/O operations in computing the updated set of frequent itemsets.

Let us assume that T_{old} is the existing database for which the set of frequent itemsets L_{old} is already computed. The respective support values of the frequent sets in L_{old} are also known. Let us assume that another set of transactions, T_{new} is now added to the database to get $T_{whole} = T_{old} \cup T_{new}$. The problem of incremental computation of frequent sets is to determine the set of all frequent sets L_{whole} of T_{whole}.

Let us assume that L_{new} is the set of all frequent sets of T_{new}. It is assumed that L_{new} is initially unknown. Then the following observations are obvious:

1. An itemset will be in L_{whole} if it is an element of both L_{new} and L_{old}.
2. Any itemset which is neither in L_{new} nor in L_{old} cannot be in L_{whole}.

Thus, if we simply start finding L_{new}, we can use condition 1 given above to determine L_{whole}, we are going to make some passes over T_{new} . Hence, we make use of these passes to compute the support of itemsets of L_{old} in T_{new}. That is, $s(X)_{Tnew}$ for X in L_{old}. But unless we read T_{old}, we may not be able find those frequent itemsets, if any, which are frequent in T_{new} as well as in T_{whole}, but infrequent in T_{old}.

Keeping the above observations in mind, let us design an incremental method. If we find $s(X)_{T_{new}}$ for all X in L_{old} (in other words, if we count the supports of frequent sets of old database with respect to the increment part of the database), and if we identify those sets which reach the σ level of support, then we can partially characterize L_{whole}. This can be achieved by just one pass over T_{new} only (supports of all the frequent itemsets can be counted in one pass). It is a partial characterization because this process takes into account the following cases:

a. The itemsets of L_{old} that are frequent in T_{new} (hence, in T_{whole}) can be determined.
b. The itemsets that belong to L_{old} but are not in L_{new} are automatically eliminated.
c. The itemsets that are neither in L_{new} nor in L_{old} are not considered.

But those itemsets that are not in L_{old} but are frequent sets in T_{whole} are not taken into account in the above step.

One way of achieving a complete characterization is to find L_{new} first and compute the supports of the itemsets in $L_{new} \setminus L_{old}$ by making one pass over T_{old}. This requires exactly one pass over the main database T_{old} and many passes over T_{new}.

This process has another disadvantage in that the database pass may not turn out to be fruitful, in the sense that there may not be any new frequent set generated by this process. In that case the database pass is unnecessary. Then the question is whether we can know in advance if any database pass is required at all or not. In other words, whether there exists a frequent set of T_{whole} which is not in L_{old}. If there is no such frequent sets, then we do not need the additional database pass.

In order to estimate this we require a new concept called the *Promoted Border Set.*

Let X be an itemset which is not frequent in T_{old}, but is frequent in T_{new} as well as in T_{whole}. All the subsets of X are frequent in T_{whole}. Since X is not frequent in L_{old}, it has a subset that is a border set in T_{old}. This border set becomes a frequent set after update. We call such a set a *promoted border.*

DEFINITION 4.8 PROMOTED BORDER

An itemset that was a border set before update and becomes a frequent set after update, is called a *promoted border* itemset.

The existence of a promoted border is an indication that we may have to read the old transaction database once again. The incremental algorithm to discover frequent sets critically depends on this observation. It is clear from the above discussion that $L_{whole} \cap (L_{new} \setminus L_{old}) \neq \emptyset$ if and only if there exists a promoted border itemset. The

correctness of this statement can be derived from the argument given in the foregoing paragraph.

However, there are few more issues to be sorted out for implementation. How do we find border sets of T_{old}? How do we determine a promoted border?

Interestingly, finding border sets is not difficult. A simple modification to the level-wise algorithm will generate the set of border itemsets along with the set of frequent sets. The idea is that while reading the database to count supports of the candidate itemsets, those itemsets having support less than σ are the border itemsets. We give below the modified algorithm with the additional step to compute border sets.

Modified *A Priori* Algorithm to Generate Border Sets and Frequent Sets

Initialize: $k := 1$, C_1 = all the 1-itemsets;
read the database to count the support of C_1 to determine L_1.
$L_1 :=$ {frequent 1-itemsets};
$k := 2$; *// k represents the pass number//*
while $(L_{k-1} \neq \emptyset)$ ***do***
begin
    $C_k :=$ ***gen_candidate_item sets*** *with the given* L_{k-1}
    prune C_k
    for all transactions $t \in T$ ***do***
    increment the count of all candidates in C_k that are contained in t ;
    $L_k := \{c \in C_k \mid s(c)_T \geq \sigma\}$;
    $B_k := \{c \in C_k \mid s(c)_T < \sigma\}$;
    $k := k + 1$;
end
    $L := \cup_k L_k$;
    $B := \cup_k B_k$;

EXAMPLE 4.2 (CONTD.)

If we use the above modification of the level-wise algorithm in Example 4.2, we get the following itemsets as the border set.

{1}→2, {9}→2, {2,5}→0, {2,6}→2, {2,7}→2, {2,8}→1,
{3,4}→1, {3,6}→0, {3,8}→0, {4,5}→1, {4,6}→1, {4,7}→2,
{4,8}→1, {5,8}→2, {6,8}→2, {5,6,7}→1.

Thus, no additional effort is needed to generate the border itemsets.

An incremental method called the *border algorithm* makes use of the concept of promoted border to avoid a fruitless pass.

4.11 BORDER ALGORITHM

The border algorithm maintains support counters for all frequent sets and all border

sets. Thus, when we get a promoted border (more precisely, when a border set is promoted to a frequent set), an additional pass over the database is made. If there is no promoted border, then the algorithm does not require even a single pass over the whole database.

The algorithm works as follows. Initially, the L_{old} and B_{old} are known along with their respective support counts. The algorithm starts by counting the supports of the itemsets of $L_{old} \cup B_{old}$ in T_{new}. This requires one pass over the increment portion of the database. During this phase, the algorithm collects two categories of itemsets: F and B.

Set F contains the itemset of L_{old} which becomes a frequent set in T_{whole}. We should take care of the threshold count while counting the support. For example, consider that $\sigma = 5\%$, $|T_{old}| = 1000$. Clearly, any itemset that is supported by 50 or more transactions in T_{old} is its frequent set. Let $|T_{new}| = 400$ and hence, $|T_{whole}| = 1400$. The itemset X is frequent in T_{whole}, if it is supported by 70 or more transactions. Thus, while determining frequent sets in T_{old}, our threshold of absolute count is 50, whereas the same in T_{whole} is 70.

Set B contains all the border sets whose support count reach the level σ and hence, are promoted border sets. If there is no promoted border, then F contains all the frequent sets of T_{whole}. But if there is at least one border set that becomes a promoted border, then the algorithm generates candidate sets which are supersets of the promoted border sets. It makes one pass over the database to count the support of these candidate sets.

Thus, the algorithm makes one pass over the increment database, and at most one pass over the whole database. We give below the formal description of the border algorithm.

Border Algorithm

read T_{new} and increment the support count of X for all $X \in L_{old} \cup B_{old}$

$F := \{X \mid X \in L_{old}$ and $s(X)_{T_{whole}} \geq \sigma\}$

$B := \{X \mid X \in B_{old}$ and $s(X)_{T_{whole}} \geq \sigma\}$

Let m be the size of the largest element in B.

Candidate-generation

for all itemsets $l_1 \in B_{k-1} \cup C_{k-1}$ ***do begin***

for all itemsets $l_2 \in B_{k-1} \cup F_{k-1} \cup C_{k-1}$ ***do begin***

    if $l_1[1] = l_2[1] \wedge l_1[2] = l_2[2] \wedge \ldots \wedge l_1[k-1] < l_2[k-1]$ ***then***

    $c = l_1[1], l_1[2] \ldots l_1[k-1], l_2[k-1]$

    $C_k = C_k \cup \{c\}$

end do

end do

Prune C_k for all k:

all subsets of $k-1$ size should be present in $B_{k-1} \cup F_{k-1} \cup C_{k-1}$

$k := k+1$

Candidate $C := \cup C_k$
read T_{whole} and count the support values of each itemset in C.
new_frequent_sets := $\{X \mid X \in C \text{ and } s(X)_{T_{whole}} \geq \sigma\}$
$L_{whole} := F \cup$ new_frequent_sets
$B_{whole} := (B_{old} \setminus B) \cup \{X \in C \text{ and } s(X)_{T_{whole}} < \sigma \text{ and all its subsets are in } L_{whole}\}$

We illustrate the working of the algorithm using the table (Table 4.1) given in Example 4.2.

EXAMPLE 4.8

Let Table 4.1 be T_{old} and add the following 5 transactions (Table 4.6) to the database. We know L_{old} and B_{old} with their respective support values.

Table 4.6 T_{new}

1	0	1	0	1	1	1	0	0
0	0	1	0	1	1	1	0	0
0	1	1	0	1	1	1	0	0
1	1	0	0	0	0	0	1	0
0	0	1	0	1	1	1	1	0

STEP 1: Read T_{new} to count supports of itemsets in L_{old} and of B_{old} and collect the itemsets which reach the σ level with respect to T_{whole} in F and B, respectively. We indicate below the itemsets that are in F and B, along with the old support and incremented support.

F:={{2}→6+2, {3}→6+4, {5}→8+4, {6}→5+4, {7}→7+4, {8}→4+2, {2,3}→3+1, {3,5}→3+4, {3,7}→3+4, {5,6}→3+4, {5,7}→5+4, {6,7}→3+4, {3,5,7}→3+4}

B:={{1}→2+2, {3,6}→0+4, {5,6,7}→1+4}

Thus, we see that we have 3 promoted border sets.

Candidate Generation We generate the candidate sets in a level-wise fashion.

Since {1} is now frequent, we need to check the supports of itemsets {1,2}, {1,3}, {1,5}, {1,6}, {1,7}, {1,8}. Note that since {4} ceases to be a frequent set after the update, we do not generate {1,4}.

For the next level of candidate-set generation, we take the all the candidate 2-itemsets generated so far and the promoted border 2-itemset, that is {1,2}. {1,3}, {1,5}, {1,6}, {1,7}, {1,8}, and {3,6} on one side. On the other side we take all the 2-itemsets in F too, that is, {2,3}, {3,5}, {3,7}, {5,6}, {5,7}, {6,7}, {1,2}. {1,3}, {1,5}, {1,6}, {1,7}, {1,8}, and {3,6}.

The candidate sets that we generate are {1,2,3}, {1,2,5}, {1,2,6}, {1,2,7}, {1,2,8}, {1,3,5}, {1,3,6}, {1,3,7}, {1,3,8}, {1,5,6}, {1,5,7}, {1,5,8}, {1,6,7}, {1,6,8}, {1,7,8}, {2,3,5}, {3,5,6}, {3,6,7}.

Similarly, the candidate 4-itemsets are {1,3,5,7} and {3,5,6,7}.

We make a single database pass.

The border algorithm was proposed by Ronen Feldman in the year 1997. Thomas *et al.* concurrently developed a similar approach for incremental discovery. Their algorithm is slightly different from the border algorithm. But both these algorithms use the same principle of determining the promoted border itemset and deciding whether a full scan is required or not. Hence, both these algorithms maintain the set of border sets along with the set of frequent sets.

One major difference in this algorithm is that it first computes L_{new}. The algorithm is described below.

Update Algorithm

compute L_{new};

for each itemset $X \in L_{old} \cup B_{old}$ ***do***
    count support of X in T_{new};

$L_{whole} := \emptyset$;

for each $X \in L_{old}$ ***do***
    if support of X in T_{whole} exceeds σ
    then $L_{whole} := L_{whole} \cup \{X\}$

for each itemset $Y \in L_{new}$ ***do***
    if $Y \notin L_{old} \wedge Y \in B_{old}$ and this support of Y in T_{whole} exceeds σ
    then $L_{whole} := L_{whole} \cup \{Y\}$;

if $L_{old} \neq L_{whole}$
then compute the border set of T_{whole}
else $B_{whole} = B_{old}$

if $L_{old} \cup B_{old} \neq L_{whole} \cup B_{whole}$
then $S = L_{whole}$
repeat
    compute $S = S \cup$ border set of S
until S does not grow
count frequency of S to determine L_{whole}
generate the border set B_{whole}

4.12 Generalized Association Rule

In the most general form, an association rule is of the form $C_1 \Rightarrow C_2$, where C_1, C_2 are conjunctions of conditions and each condition is either $l_i = v$; or $l_i \in [low, upper]$ for each item l_i in A, where v, *low* and *upper* are some values in the domain of l_i. In this framework, the association rule that we have studied so far becomes a special case where $C = [l_i =$ 'yes'$]$. Databases in the real world usually have numeric values such as

age and balance of account. Thus, it is important to find association rules for numeric and categorical attributes.

Another feature of association rule generators is an *item hierarchy*. In our example, we were using generic terms: diapers, and beer. Let us assume that stores sell specific items: 6d beer in a 31b box, 24 oz diapers product ID 32225, etc. Normally, rules that are somewhat more general, like the rules we have been using as examples, are frequently what is expected. But in some cases, one might be interested in rules that deal in specific items or we might be interested in rules that differentiate between, say, beers sold in boxes vs beer sold in bulk. A product hierarchy can be used to structure a hierarchy of items that permits grouping at different levels.

Note also that in an association rule time does not play a part. Contrast this with a sequencing rule in which time is important, and that might read: W*hen people buy diapers they also buy beer in the time period 2 to 4 months later, 18% of the time.* We shall discuss this problem in Chapter 9.

Association rules, and the support-confidence framework used to mine them, are well-suited to the market–basket problem. Other basket data problems, while seemingly similar, have requirements that the support-confidence framework does not support. For example, negative implications of the type: "*when people buy batteries, they do not usually also buy cat food.*" While perhaps not so useful to the marketing staff of supermarkets, such implications can be helpful in many other settings. For example, fire code inspectors trying to mine useful fire prevention measures might like to know of any negative correlations between certain types of electrical wiring and the occurrence of fires.

The bigger problem is that the support-confidence framework does not work well when correlation is the appropriate measure.

Example 4.9

Suppose we have market–basket data from a grocery store, consisting of n baskets. Let us focus on the purchase of tea and coffee. In the following table (Table 4.7), rows t and $\bar{t}$ correspond to baskets that do and do not, respectively, contain tea, and similarly columns c and $\bar{c}$ correspond to coffee. The numbers represent the percentage of baskets.

Let us apply the support-confidence framework to the potential association rule, $t \Rightarrow c$. The support for this rule is 20%, which is fairly high. The confidence is defined to be the conditional probability that a customer buys coffee, given that she buys tea, i.e., 20/25 = 0.8 or 80%, which too is pretty high. At this point, we may conclude that the rule $t \Rightarrow c$ is a valid rule.

Table 4.7

	c	$\bar{c}$
t	20	5
$\bar{t}$	70	5

However, consider now the fact that the *a priori* probability that a customer buys coffee is 90%. In other words, a customer who is known to buy tea is less likely to buy coffee (by 10%) than a customer about whom we have no information. Of course, it may still be interesting to know that such a large number of people who buy tea also buy coffee, but stating that rule by itself is at best incomplete information and at worst misleading. The truth here is that there is a negative correlation between buying tea and buying coffee; at least that information should be provided along with the association rule $t \Rightarrow c$. One way to deal with this situation is to identify also the rules $\bar{c} \Rightarrow t$, $\bar{t} \Rightarrow c$ and $c \Rightarrow t$. Another way of measuring correlation is to compute $P[t \wedge c]/(P[t] \times P[c])$. The fact that this quantity is significantly less than 1 indicates a negative correlation, since the numerator is the actual likelihood of seeing a customer purchase both tea and coffee, and the denominator is what the likelihood would have been in the case when the two purchases are completely independent.

4.13 Association Rules with Item Constraints

In practice, the users are often interested only in a subset of associations, for instance, those containing at least one item from a user-defined subset of items. A straightforward way of doing this is to carry out a filtering as a post-processing step after any association rule mining algorithm is invoked. However, it would be desirable to incorporate these constraints within the steps of the association discovery algorithm.

Let B be a Boolean expression over the set of itemsets A. We assume, without loss of generality, that B is in disjunctive normal form (DNF). That is, B is of the form, $D_1 \vee D_2 \vee \ldots \vee D_m$, where each disjunct D_i is of the form $\alpha_{i1} \wedge \alpha_{i2} \ldots \wedge \alpha_{in}$. Each of the α refers either to the presence of an item, l_{ij} or to the absence of an item, $\neg l_{ij}$.

The problem of the mining association rule with item-constraint B is to discover all rules that satisfy B and have support and confidence greater than or equal to the user-specified minimum support and minimum confidence, respectively.

We can split the problem into three phases.

Phase 1

Find all frequent itemsets that satisfy the Boolean expression, B. Remember that there are two types of operations used for this problem: candidate-set generation and counting support. The techniques for counting support remains unchanged for the current problem. However, the earlier method of candidate-set generation may not assure that the frequent sets are those which satisfy the Boolean constraints.

This subproblem is tackled in the following way:

1. Generate a set of selected items S, such that any itemset that satisfies B will contain at least one selected item.

2. Modify the candidate-generation procedure to generate only candidates that contain selected items.
3. Discard frequent itemsets that do not satisfy B.

PHASE 2

To generate rules from these frequent itemsets, we need to find the support of all subsets of frequent itemsets that do not satisfy B. Recall that to generate a rule ABCD, we need the support of AB to find the confidence of the rule. However, AB may not satisfy B and hence may not have been counted in Phase 1. So we generate all subsets of the frequent itemsets found in Phase 1, then make an extra pass over the data set to count the support of those subsets that are not present in the output of Phase 1.

PHASE 3

To generate rules from the frequent itemsets found in Phase 1, using the frequent itemsets found in Phase 1 and 2 to compute confidence.

GENERATING SELECTED ITEMS

We can generate S by choosing one element α_{ij} from each disjunct D_i in S and adding either l_{ij} or all the elements in $A-\{l_{ij}\}$ to S, based on whether α_{ij} is l_{ij} or $\neg l_{ij}$, respectively. We define an element $\alpha_{ij} = l_{ij}$ in B to be present in S if l_{ij} is in S, and element $\alpha_{ij} = \neg l_{ij}$ to be present if all the items in $A-\{l_{ij}\}$ are in S.

4.14 SUMMARY

In this chapter, an attempt is made to summarize all the major techniques of discovering associations rules for large databases. The chapter begins with establishing the foundational concepts on this subject and discusses the algorithms like *A priori*, Partition, Pincer-Search, Dynamic Itemset counting and FP-tree Growth. It also addresses related problems like incremental computation of association rules. It summarizes the techniques of generalized association rule discovery and algorithms for association rules with item-constraints.

FURTHER READING

A detailed bibliography on the discovery of association rules is given below. Lin and Kedem [Lin, 1998] proposed the Pincer-search algorithm. Dynamic Itemset mining was proposed by Brin *et al.* in [Brin, 1997]. The FP-tree algorithm was presented by Han *et al.* [Han, 200]. Shenoy and others proposed a vertical mining algorithm VIPER in [Shenoy, 2000]. The hash-based technique was initially proposed by Park in [Park,

1997]. Recently, Zaki [Zaki, 2000] presented a method of generating non-redundant association rules. Zaki and Ogihara in [Zaki, 1998] and Gunopulos *et al.* [Gunopulos, 1997] provide the theoretical foundation for association rules and frequent sets.

EXERCISES

1. Define a frequent set. Define an association rule.
2. Define a border set. Show that every subset of any itemset must contain either a frequent set or a border set.
3. Discuss the importance of discovering association rules.
4. The discovery of the association rule can be formulated in terms of relational algebraic operations. Comment.
5. Discuss the concepts of frequent sets, confidence and support.
6. Describe the working of the pincer-search algorithm.
7. The pincer-search algorithm is a bi-directional search, whereas the level-wise algorithm is a unidirectional search. Comment.
8. The pincer-search algorithm finds only maximal frequent sets. Comment.
9. Describe the principle of pruning in level-wise algorithms. The set of all frequent sets can be generated even by ignoring the pruning step. Comment.
10. Describe the candidate-generating step of level-wise algorithms. What is its importance?
11. Describe the working dynamic itemset counting technique. When do you move itemsets from dashed structures to solid structures.
12. What is the incremental discovery of an association rules? What are the algorithms for incremental discovery? Discuss the important features of the algorithm.
13. What is the problem of association rule with item-constraints? Propose a method for this.
14. Define a FP-tree. Discuss the method of computing a FP-tree.

BIBLIOGRAPHY

Aggarwal Charu, and Yu Philip. Mining large itemsets for association rules. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering,* **21**, no. **1**, March 1998.

Agrawal R., Imielinski T., and Swami A. Mining associations between sets of items in massive databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Washington D.C., May 1993, pp. 207–216.

Agrawal R., Mannila H., Srikant R., Toivonen H., and Verkamo A.I. Fast discovery of

association rules. *Advances in Knowledge Discovery and Data Mining,* Chapter 12, AAAI/MIT Press, 1995.

Agrawal R., and Shafer J.C. Parallel mining of association rules: Design, implementation and experience. *IEEE Transactions on Knowledge Data and Engineering*, December 1996, **8(6)**: pp. 962–969.

Agrawal R., and Srikant. R. Fast algorithms for mining association rules. *Proceedings of the 20th International Conference on Very Large Databases,* Santiago.

Alsabti K., Ranka S., and Singh V. A one-pass algorithm for accurately estimating quantiles for disk-resident data. In *Proceedings of VLDB'97.*

Amir R., Feldman, and Kashi R. A new and versatile method for association generation. *Principles of Data Mining and Knowledge Discovery*, First European Symposium, PKDD'97, Komorowski and Zytkow (eds.), Springer LNAI 1263, Trondheim, Norway, 1997, pp. 221–231.

Ayan N.F., Tansel A.U., and Arkun E. An efficient algorithm to update large itemsets with early pruning. In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining (KDD '99)*, San Diego, California, USA, August 1999.

Bayardo R.J. Jr., Agrawal R., and Gunopulos D. Constraint-based rule mining in large, dense databases. In *Proceedings of the 15th International Conference on Data Engineering*, Sydney, Australia, March 1999.

Borges Jose, and Levene Mark. Mining association rules in hypertext databases. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD-98)*, New York City, August 1998.

Brin Sergey, Motwani Rajeev, Tsur Dick, and Ullman Jeffrey. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of 1997 ACM SIGMOD*, Montreal, Canada, June 1997.

Brin Sergey, Motwani Rajeev, and Silverstein Craig. Beyond market baskets: Generalizing association rules to correlations. In *Proceedings of 1997 ACM SIGMOD*, Montreal, Canada, June 1997.

Fukuda T., Morimoto Y., Morishita S., and Tokuyama T. Mining optimized association rules for numeric attributes. In *Proceedings of the 15th ACM SIGACT- SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '96)*, Montreal, Canada, June 1996.

(Sam) Han Eui-Hong, Karyapis George, and Kumar Vipin. Scalable parallel data mining for association rules. In *Proceedings of 1997 ACM-SIGMOD International Conference on Management of Data*, May 1997.

Han J., Pei J., and Yin Y. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM-SIGMOD International Conference on Management of Data*, Dallas, Texas, USA, May 2000.

Hidber C. Online association rule mining. In *Proceedings of the 1999 ACM SIGMOD Conference on Management of Data*, 1999.

Hipp J., Myka A., Wirth R., and Guntzer U. A new algorithm for faster mining of generalized association rules. In *Proceedings of the 2nd European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD '98)*, Nantes, France, September 1998.

Hipp J., Guntzer U., and Nakhaeizadeh G. Mining association rules: Deriving a superior algorithm by analysing today's approaches. In *Proceedings of the 4th European Conference on Principles and Practice of Knowledge Discovery*, Lyon, France, September 2000.

Houtsma M., and Swami A. Set-oriented mining for association rules in relational databases. *Technical Report RJ 9567*, IBM Almaden Research Center, San Jose, California, October 1993.

Klemettinen M., Mannila H., Ronkainen P., Toivonen H., and Verkamo A.I. Finding interesting rules from large sets of discovered association rules. In *Proceedings of the 3rd International Conference on Information and Knowledge Management*, Gaithersburg, Maryland, 29 Nov - 2 Dec 1994.

Lin D-I., and Kedem Z.M. Pincer-search: A new algorithm for discovering maximum frequent set. In *Sixth International Conference on Extending Database Technology*, March 1998.

Mannila H., and Toivonen H. Multiple uses of frequent sets and condensed representations. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*, Portland, Oregon, August 1996, pp. 189–194.

Motwani R., Cohen E., Datar M., Fujiware S., Gionis A., Indyk P., Ullman J.D., and Yang C. Finding interesting associations without support pruning. In *Proceedings of the 16th International Conference on Data Engineering (ICDE)*, IEEE, 2000.

Ng R., Lakshmanan L.V.S., Han J., and Pang A. Exploratory mining and pruning optimizations of constrained association rules. In *Proceedings of the 1998 ACM-SIGMOD Conference on Management of Data*, Seattle, Washington, June 1998.

Park Jong Soo, Chen Ming-Syan, and Yu Philip S. Using a hash-based method with transaction trimming for mining association rules. *IEEE Transactions on Knowledge and Data Engineering*, **9**, no. 5, Sept/Oct 1997, pp. 813–825.

Pasquier N., Bastide Y., Taouil R., and Lakhal L. Discovering frequent closed itemsets for association rules. In *Proceedings of the 7th International Conference on Database Theory (ICDT'99)*, Jerusalem, Israel, January 1999.

Pei J., Han J., and Mao R. An efficient algorithm for mining frequent closed itemsets. In *Proceedings of the 2000 ACM-SIGMOD International Conference on Management of Data*, Dallas, Texas, USA, May 2000.

Rastogi R., and Shim K. Mining optimized support rules for numeric attributes. In *Proceedings of the 15th International Conference on Data Engineering*, IEEE Computer Society Press, Sydney, Australia, March 1999.

Savasere, E. Omiecinski, and Navathe S. An efficient algorithm for mining association rules in large databases. In *Proceedings of the 21st Conference on Very Large Databases (VLDB '95)*, Zurich, Switzerland, September 1995.

Shintani T., and Kitsuregawa M. Hash-based parallel algorithm for mining association rules. In *Proceedings of IEEE Fourth International Conference on Parallel and Distributed Information Systems*, 1996, pp.19–30.

Silverstein C., Brin S., Motwani R., and Ullman J.D. Scalable techniques for mining causal structures. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data,* Seattle, Washington, USA, June 1998.

Srikant R., Vu Q., and Agrawal R. Mining association rules with item constraints. In *Proceedings of the 3rd International Conference on Knowledge Discovery in Databases and Data Mining,* Newport Beach, California, August 1997.

Srikant R., and Agrawal R. Mining generalized association rules. In *Proceedings of the 21st International Conference on Very Large Databases*, Zurich, Switzerland.

Thomas S., Bodagala S., Alsabti K., and Ranka S. An efficient algorithm for the incremental updation of association rules in large databases. In *Proceedings of the 3rd International Conference on KDD and Data Mining (KDD'97)*, Newport Beach, California, August 1997.

Toivonen H., Klemettinen M., Ronkainen P., Hatonen K., and Mannila H. Pruning and grouping discovered association rules. In *MLnet Workshop on Statistics, Machine Learning, and Discovery in Databases*, Heraklion, Crete, Greece, April 1995, pp. 47–52.

Tsur Dick, Ullman Jeffrey, Clifton Chris, Abiteboul Serge, Motwani Rajeev, Nestorov Svetlozar, and Rosenthal Arnie. Query flocks: a generalization of association-rule mining. In *Proceedings of the 1998 ACM SIGMOD*, Seattle.

Wang K., Tay W., and Liu B. Interestingness-based interval merger for numeric association rules. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD-98)*, New York City, August 1998.

Yoda K., Fukuda T., Morimoto Y., Morishita S., and Tokuyama T. Computing optimized rectilinear regions for association rules. In *Proceedings of the 3rd Conference on Knowledge Discovery and Data Mining (KDD'97)*, pp. 96–103, Los Angels, August 1997.

Zaki Mohammed, Ogihara Mitsunori, Parthasarathy Srinivasan, and Li Wei. Parallel data mining for association rules on shared-memory multi- processors. In *Supercomputing'96*, Pittsburg, PA, Nov 17–22, 1996.

Zaki Mohammed, Parthasarathy Srinivasan, Ogihara Mitsunori, and Li Wei. New algorithms for fast discovery of association rules. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining* (KDD'97), Newport Beach, California, August, 1997, pp 283–286.

Zaki Mohammed, Parthasarathy Srinivasan, and Li. Wei. A localized algorithm for parallel association mining. In *9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, Newport, Rhode Island, pp 321–330.

Zaki Mohammed J., and Ogihara Mitsunori. Theoretical foundations of association rules. In *3rd SIGMOD'98 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD)*, Seattle, WA, June 1998.

5

CLUSTERING TECHNIQUES

5.1 INTRODUCTION

Clustering is a useful technique for the discovery of data distribution and patterns in the underlying data. The goal of clustering is to discover both the dense and the sparse regions in a data set. Data clustering has been studied in the statistics, machine learning, and database communities with diverse emphases. The earlier approaches do not adequately consider the fact that the data set can be too large to fit in the main memory. In particular, they do not recognize that the problem must be viewed in terms of how to work with limited resources. The main emphasis has been to cluster with as high an accuracy as possible, while keeping the I/O costs high. Thus, it is not relevant to apply the classical clustering algorithms in the context of data mining and it is necessary to investigate the principle of clustering to devise

efficient algorithms which meet the specific requirement of minimizing the I/O operations. It is also to be noted that the basic principle of clustering hinges on a concept of distance metric or similarity metric. Since the data are invariably real numbers for statistical applications and pattern recognition, a large class of metrics exists and one can define one's own metric depending on a specific requirement. But in databases, there exist objects which cannot be ordered, and their attributes can be categorical in nature. For example, consider a market–basket database. Typically, the number of items, and thus the number of attributes in such a database is very large, while the size of an average transaction is much smaller. Furthermore, customers with similar buying patterns, who belong to a single cluster, may buy a small subset of items from a much larger set that defines the cluster. Thus, conventional clustering methods that handle only numeric data are not suitable for data mining purposes.

This chapter deals with different clustering techniques for data mining. In the last five years, there has been an increasing interest in devising efficient clustering algorithms for data mining. In a short span of 2 to 3 years a very large number of new and interesting algorithms have been proposed. The aim of this chapter is to survey these algorithms and present them in a form which will be useful to the research community for future research. We shall first discuss different approaches of clustering of numerical data and then we shall discuss the recent algorithms of clustering categorical data. The clustering algorithms for numerical data are again categorized into two classes: partition and hierarchical.

In Section 5.2, we shall briefly outline different approaches to clustering. Section 5.3 discusses the underlying principles of partitioning algorithms. Three different partitioning algorithms, namely PAM, CLARA and CLARANS, are discussed in Sections 5.4, 5.5 and 5.6, respectively. Section 5.7 introduces the concepts of hierarchical clustering algorithms. In this category, we discuss DBSCAN in Section 5.8; CURE in Section 5.9; and BIRCH in Section 5.10. The conventional mining algorithms handle only numeric data, while clustering techniques for the purpose of data mining are expected to handle categorical data also. We shall discuss the principles of clustering of categorical data in Section 5.11. Three algorithms in this category that are discussed in detail are STIRR, ROCK and CACTUS in Sections 5.12–5.14.

5.2 Clustering Paradigms

There are two main approaches to clustering—*hierarchical clustering* and *partitioning clustering*. Besides, clustering algorithms differ among themselves in their ability to handle different types of attributes, numeric and categorical, in accuracy of clustering, and in their ability to handle disk-resident data.

Hierarchical vs Partitioning

The partition clustering techniques partition the database into a predefined number of clusters. They attempt to determine k partitions that optimize a certain criterion function. The partition clustering algorithms are of two types: k-means algorithms and k-medoid algorithms. Another type of algorithms can be k-mode algorithms.

The hierarchical clustering techniques do a sequence of partitions, in which each partition is nested into the next partition in the sequence. It creates a hierarchy of clusters from small to big or big to small. The hierarchical techniques are of two types—*agglomerative* and *divisive* clustering techniques. *Agglomerative clustering* techniques start with as many clusters as there are records, with each cluster having only one record. Then pairs of clusters are successively merged until the numbers of clusters reduces to k. At each stage, the pairs of the clusters that are merged are the ones nearest to each other. If the merging is continued, it terminates in a hierarchy of clusters which is built with just a single cluster containing all the records, at the top of the hierarchy. *Divisive clustering* techniques take the opposite approach from agglomerative techniques. This starts with all the records in one cluster, and then try to split that cluster into small pieces.

Numeric vs Categorical

Clustering can be performed on both numerical data and categorical data. For the clustering of numerical data, the inherent geometric properties can be used to define the distances between the points. But for clustering of categorical data, such a criterion does not exist and many data sets also consist of categorical attributes, on which distance functions are not naturally defined. Recently, the problem of clustering categorical data has attracted interest. As an example, consider the MUSHROOM data set in the popular UCI machine-learning repository. Each tuple in the data set describes a sample of gilled mushrooms using twenty-two categorical attributes. For instance, the cap colour attribute can take values from the domain {brown, buff, cinnamon, gray, green, pink, purple, red, white, yellow}. It is hard to reason that one colour is less than another colour in a way similar to real numbers. It is also hard to determine an ordering or to quantify the dissimilarity among such attributes.

5.3 Partitioning Algorithms

Partitioning algorithms construct partitions of a database of N objects into a set of k clusters. The construction involves determining the optimal partition with respect to an objective function. There are approximately $kN/k!$ ways of partitioning a set of N data points into k subsets. An exhaustive enumeration method that can find the global

optimal partition, is practically infeasible except when N and k are very small. The partitioning clustering algorithm usually adopts the *Iterative Optimization* paradigm. It starts with an initial partition and uses an iterative control strategy. It tries swapping data points to see if such a swapping improves the quality of clustering. When swapping does not yield any improvements in clustering, it finds a locally optimal partition. This quality of clustering is very sensitive to the initially selected partition. There are the two main categories of partitioning algorithms. They are

(i) k-means algorithms, where each cluster is represented by the center of gravity of the cluster.
(ii) k-medoid algorithms, where each cluster is represented by one of the objects of the cluster located near the center.

Most of special clustering algorithms designed for data mining are k-medoid algorithms.

5.4 K-MEDOID ALGORITHMS

PAM

PAM (**P**artition **A**round **M**edoids) uses a k-medoid method to identify the clusters. PAM selects k objects arbitrarily from the data as medoids. Each of these k objects are representatives of k classes. Other objects in the database are classified based on their distances to these k-medoids (we say that the database is partitioned with respect to the selected set of k-medoids). The algorithm starts with arbitrarily selected k-medoids and iteratively improves upon this selection. In each step, a swap between a selected object O_i and a non-selected object O_h is made, as long as such a swap results in an improvement in the quality of clustering. To calculate the effect of such a swap between O_i and O_h a cost C_{ih} is computed, which is related to the quality of partitioning the non-selected objects to k clusters represented by the medoids. The algorithm has two important modules—the partitioning of the database for a given set of medoids and the iterative selection of medoids.

PARTITIONING

If O_j is a non-selected object and O_i is a medoid, we then say that O_j belongs to the cluster represented by O_i, if $d(O_i, O_j) = \text{Min}_{Oe}\, d(O_j, O_e)$, where the minimum is taken over all medoids O_e and $d(O_a, O_b)$ determines the distance, or dissimilarity, between objects O_a and O_b. The dissimilarity matrix is known prior to the commencement of PAM. The quality of clustering is measured by the average dissimilarity between an object and the medoid of the cluster to which the object belongs.

Iterative Selection of Medoids

Let us assume that $O_1, O_2, ..., O_k$ are k-medoids selected at any stage. We denote $C_1, C_2, ..., C_k$ as the respective clusters. From the foregoing discussion, for a non-selected object $O_j, j \neq 1, 2, ..., k$

if $O_j \in C_m$ then $\text{Min}_{(1 \leq i \leq k)}\, d(O_j, O_i) = d(O_j, O_m)$.

Let us now analyze the effect of swapping O_i and O_h. In other words, let us assume that an unselected object O_h becomes a medoid replacing an existing medoid O_i. Thus, the new set of k-medoids is $Kmed' = \{O_1, O_2, ..., O_{i-1}, O_h, O_{i+1}, ..., O_k\}$, where O_h replaces O_i as one of the medoids from $Kmed = \{O_1, O_2, ..., O_k\}$. We shall now compare the quality of clustering. Due to the change in the set of medoids, there will be three types of changes that can occur in the actual clustering. For convenience, let us denote C_h as the new clusters with O_h as the representative.

1. A non-selected object O_j, such that $O_j \in C_i$ before swapping and $O_j \in C_h$ after swapping. This case arises when the following conditions hold:

   Min $d(O_j, O_e) = d(O_j, O_i)$,

   minimum taken over O_e in *Kmed.*

   $\text{Min}_{e \neq i}\, d(O_j, O_e) = d(O_j, O_h)$,

   minimum taken for all O_e in $Kmed'$.

   Let us define a cost for this change as

   $C_{jih} = d(O_j, O_h) - d(O_j, O_i)$

2. A non-selected object $O_j \in C_i$ before swapping and $O_j \in C_g$, $g \neq h$, after swapping. This case arises when

   Min $d(O_j, O_e) = d(O_j, O_i)$,

   minimum taken over O_e in *Kmed.*

   and Min $d(O_j, O_e) = d(O_j, O_g)$, $g \neq h$.

   minimum taken over O_e in $Kmed'$.

   Define a cost as $C_{jih} = d(O_j, O_g) - d(O_j, O_i)$

3. A non-selected object $O_j \in C_g$ before swapping and $O_j \in C_h$ after swapping. This case arises when

   Min $d(O_j, O_e) = d(O_j, O_g)$,

   minimum taken over O_e in *Kmed.*

and Min $d(O_j, O_e) = d(O_j, O_h)$,

minimum taken over O_e in $Kmed'$.

Define a cost for this case as,

$$C_{jih} = d(O_j, O_h) - d(O_j, O_g)$$

Please note that for any non-selected object O_j, only one of these three cases can occur. The total cost of swapping two medoids is the sum of such costs for all non-selected objects.

The following figures (Figures 5.1 and 5.2) illustrate the working of the swapping in PAM. There are 12 objects which are required to be classified into 4 clusters.

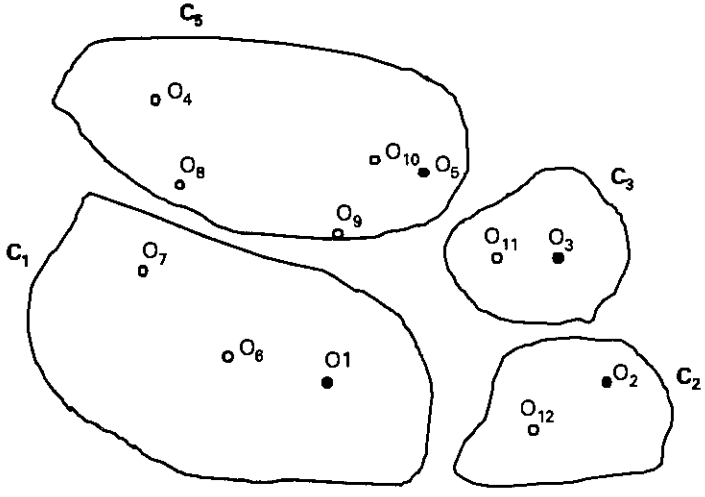


Figure 5.1 Partitioning Before Swapping

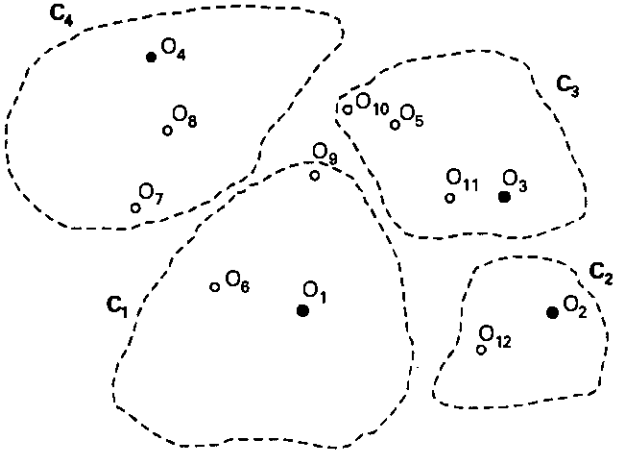


Figure 5.2 Clustering After Swapping O_5 by O_4

Initially, the selected medoids are O_1, O_2, O_3 and O_5. Based on closest distance, the remaining objects are classified as shown in the figures. Let us assume that O_4 is introduced as a medoid in place of O_5. For convenience, we denote d_{ij} for $d(O_i, O_j)$ and the distance metric is symmetric. The three cases are illustrated below.

Case 1 The object O_8 which was in the cluster C_5 before swapping is now in the cluster C_4 after swapping.

Hence, the cost $C_{854} = d_{58} - d_{48}$.

Case 2 The object O_9 was in C_5 before swapping and is now in C_1 after swapping.

The cost for O_9 is $C_{954} = d_{19} - d_{59}$.

Case 3 The object O_7 was in the cluster C_1 before swapping and is in C_4 after swapping.

Thus, the cost is $C_{754} = d_{47} - d_{17}$.

Define the total cost of swapping O_i and O_h as

$$C_{ih} = \sum_j C_{jih},$$

where the sum is taken for all objects j.

In the above example, the sum is taken over all the 12 objects to compute C_{54}.

If C_{ih} is negative, then the quality of clustering is improved by making O_h a medoid in the place of O_i. PAM attempts to compute C_{ih} for all selected objects O_i and all non-selected objects O_h. The process is repeated until we cannot find a negative C_{ih}. If more than one pair of (i, h) with negative costs are found, then the pair corresponding to the most negative C_{ih} is chosen for swapping. The algorithm can be formally stated as follows:

PAM Algorithm

Input: Database of objects D.
select arbitrarily k representative objects, *Kmed*.
mark these objects as "selected" and mark the remaining as "non-selected".
    do for all selected object O_i
        do for all non-selected objects O_h
            compute C_{ih}
        end do.
    end do.

select i_{min}, h_{min} such that $C_{imin,hmin} = \text{Min}_{i,h}\, C_{ih}$
    if $C_{imin,hmin} < 0$
        then swap: mark O_i as non-selected and O_h as selected.
    repeat
find clusters $C_1, C_2, C_3, \ldots, C_k$.

PAM is very robust to the existence of outliers. The clusters found by this method do not depend on the order in which the objects are examined. However, it cannot handle very large volumes of data.

5.5 CLARA

It can be observed that the major computational efforts for PAM are to determine *k*-medoids through an iterative optimization. Though CLARA follows the same principle, it attempts to reduce the computational effort. Instead of finding representative objects for the entire data set, CLARA draws a sample of the data set, and applies PAM on this sample to determine the optimal set of medoids from the sample. It then classifies the remaining objects using the partitioning principle. If the sample were drawn in a sufficiently random way, the medoids of the sample would approximate the medoids of the entire data set. The steps of CLARA are summarized below.

CLARA Algorithm

Input: Database of D objects.
repeat for m times
    draw a sample $S \subseteq D$ randomly from D.
    call PAM (S, k) to get k medoids.
    classify the entire data set D to $C_1, C_2 \ldots C_k$.
    calculate the quality of clustering as the average dissimilarity.
end.

5.6 CLARANS

CLARANS (**C**lustering **L**arge **A**pplications based on **Ran**domized **S**earch) is similar to PAM and CLARA, but it applies a randomized Iterative-Optimization for the determination of medoids. It is easy to see that in PAM, at every iteration we examine $k(N\text{-}k)$ swaps to determine the pair corresponding to the minimum cost. On the other hand, CLARA tries to examine fewer elements by restricting its search to a smaller sample of the database. Thus, if the sample size is $S \leq N$, it examines at most $k(S\text{-}k)$ pairs at every iteration. Thus, in CLARA, an object can be a medoid only if it is in the randomly selected sample. But CLARANS does not restrict the search to any particular subset of objects. Neither does it search the entire data set. It begins with PAM and randomly selects few pairs (i, h), instead of examining all pairs, for swapping at the current state. CLARANS, like PAM, starts with a randomly selected set of k-medoids. It checks at most the "*maxneighbor*" number of pairs for swapping

and, if a pair with negative cost is found, it updates the medoid set and continues. Otherwise, it records the current selection of medoids as a local optimum and restarts with a new randomly selected medoid, set to search for another local optimum. CLARANS stops after the "*numlocal*" number of local optimal medoid sets are determined, and returns the best among these.

CLARANS Algorithm

```
Input (D, k, maxneighbor and numlocal)
select arbitrarily k representative objects.
mark these objects as "selected" and all other objects as non-selected. Call it current.
set e = 1.
do while (e ≤ numlocal)
    set j = 1
    do while (m ≤ maxneighbor )
        select randomly a pair ( j, h) such that O_j is a selected object and O_h is a non-selected object.
        compute the cost C_jh.
        if C_jh is negative
            "update current"
            mark O_j non-selected, O_h selected and m = 1
        else
            increment m ← m + 1
    end do
    compare the cost of clustering with "mincost"
    if current_cost < mincost
        mincost ← current_cost
        best_node ← current
    increment e ← e + 1
end do
return "best node"
```

CLARANS is a medoid-based method, which is more efficient than the earlier medoid-based methods, but suffers from two major drawbacks: it assumes that all objects fit into the main memory, and the result is very sensitive to input order. In addition, it may not find a real local minimum due to the trimming of its searching, controlled by '*maxneighbor*'. Focussing techniques is a way to improve CLARANS's ability to deal with disk-resident data sets, by focussing only on relevant parts of the database using R* trees.

5.7 Hierarchical Clustering

Hierarchical algorithms create a hierarchical decomposition of the database. The algorithms iteratively split the database into smaller subsets, until some termination condition is satisfied. The hierarchical algorithms do not need k as an input parameter,

which is an obvious advantage over partitioning algorithms. However, the disadvantage of the hierarchical algorithm is that the termination condition is to be specified. The hierarchical decomposition can be represented as a dendogram in two ways:

(i) Bottom-up (Agglomerative) approach, and
(ii) Top-down (Divisive) approach.

The basic agglomerative, hierarchical clustering algorithm works in the following way. Initially, each object is placed in a unique cluster. For each pair of clusters, some value of dissimilarity or distance is computed. For instance, the distance may be the minimum distance of all pairs of points from the two clusters. At every step, the clusters with the minimum distances in the current clustering are merged, until the whole data set forms a single cluster. One can set the termination criteria by fixing the critical distance D_{min} between the clusters.

5.8 DBSCAN

DBSCAN (Density Based Spatial Clustering of Applications of Noise) uses a density-based notion of clusters to discover clusters of arbitrary shapes. The key idea of DBSCAN is that, for each object of a cluster, the neighbourhood of a given radius has to contain at least a minimum number of data objects. In other words, the density of the neighbourhood must exceed a threshold. The critical parameter here is the distance function for the data objects. The following concepts are introduced here in the context of DBSCAN.

DEFINITION 5.1 ε-NEIGHBOURHOOD OF AN OBJECT

For a given non-negative value ε, the *ε-neighbourhood* of an object O_i, denoted by $N_\varepsilon(O_i)$, is defined by $N_\varepsilon(O_i) = \{O_j \in D \mid d(O_i, O_j) \leq \varepsilon\}$

DEFINITION 5.2 CORE OBJECT

An object is said to be a *Core Object* if $| N_\varepsilon(O) | \geq$ MinPts. A core object is an object which has a neighbourhood of user-specified minimum density.

DEFINITION 5.3 DIRECTLY-DENSITY-REACHABLE

An object O_i is *directly-density-reachable* from an object O_j with respect to ε and Minpts, if O_j is a core object and O_i is in its ε-neighbourhood.

1. $O_i \in N_\varepsilon(O_j)$
2. $| N_\varepsilon(O_j) | \geq$ MinPts. (In other words, O_j is a core object.)

DEFINITION 5.4 DENSITY-REACHABLE

An object O_i is *density-reachable* from an object O_j with respect to ε and MinPts in D if there is a chain of objects $O_1, O_2, \ldots, O_n$, such that $O_1=O_j$, $O_n=O_i$, such that $O_e \in D$ and O_{e+1} is directly-density-reachable from O_e with respect to ε and MinPts in D.

Generally, the density-reachability relation is transitive but not symmetric. If O_i is density-reachable from O_j, then O_j is a core object, but O_i need not be a core object. Thus, we cannot say that O_j is also density-reachable from O_i. The directly-density-reachable relation is a special case of the density-reachable relation.

DEFINITION 5.5 DENSITY-CONNECTED

An object O_i is *density-connected* to an object O_j with respect to ε and MinPts in D if there is another object $O \in D$, such that both O_i and O_j are density-reachable from O, with respect to ε and MinPts in D.

The density-connectivity relation is a symmetric relation and within the set of density-reachable objects, the relation is reflexive too.

DEFINITION 5.6 CLUSTER

A *Cluster* C with respect to ε and MinPts is a non-empty subset of D satisfying the following conditions:

- For all $O_i, O_j \in D$, if $O_i \in C$ and O_j is density-reachable from O_i with respect to ε and MinPts, then $O_j \in C$.
- For all $O_i, O_j \in C$, O_i is density connected to O_j with respect to ε and MinPts.

DEFINITION 5.7 NOISE

Let $C_1, C_2, \ldots, C_k$ be the clusters of D with respect to ε and MinPts. Then, we define the noise as the set of objects in D which do not belong to any cluster C_i as *Noise* = $\{O \in D \mid \forall i, O \notin C_i\}$.

We identify two different kinds of objects in a DBSCAN—*core objects* and *non-core objects*. Non-core objects, in turn, are either border objects or noise objects. Two border objects are possibly not density-reachable from each other. However, a border object is always density-reachable from a core object. A noise object is a non-core object, which is not density-reachable from other core objects. DBSCAN's procedure for finding a cluster is based on the fact that a cluster is uniquely determined by any of its core objects. It can be seen that

1. Given an arbitrary object O_i for which the core object condition holds, the set $\{O \in D$: O is density reachable from $O_i\}$ forms a complete cluster C and $O_i \in C$.
2. Given a cluster C and an arbitrary core object $O_i \in C$, C in turn equals the set of all objects which are density-reachable from O_i in D.

The DBSCAN algorithm maintains the set of objects in three different categories. These are: classified, unclassified and noise. Each classified objects has an associated

cluster-id, indicating the cluster in which it is included. A noise object may also have an associated dummy cluster-id. For both classified and noise objects, the ε-neighbourhoods are already computed. The unclassified category of objects do not have any cluster-id and their neighbourhoods are not computed. The algorithm gradually converts an unclassified object into a classified or a noise object.

At every step, the algorithm starts with an unclassified object and a new cluster-id associated with it. We study its ε-neighbourhood to see if the neighbourhood is adequately dense or not. If its density does not exceed the threshold MinPts, then it is marked as a noise object. Otherwise, all the objects that are within its ε-neighbourhood are retrieved and put into a list of candidate objects. These objects may be either unclassified or noise. Note that the neighbourhood cannot contain a classified object. If the object is a noise object, the current cluster-id is assigned to it. If the object is unclassified, then the current cluster-id is assigned to it and it is included in the list of candidate objects for which the ε-neighbourhoods are to be obtained. The algorithm continues till the list of candidate objects is empty. Thus, one cluster with the given cluster-id is determined. The algorithm repeats this process for other unclassified objects and terminates when all the objects are marked as either classified or noise.

We present the algorithm below.

DBSCAN Algorithm

Algorithm DBSCAN (D, ε, MinPts)
Input: Database of objects D
do for all $O \in D$
    if O is unclassified
        call function **expand_cluster**(O, D, ε, MinPts)
end do.

Function expand_cluster (O, D, ε, MinPts):
get the ε-neighbourhood of O as $N_\varepsilon(O)$
    if $|N_\varepsilon(O)| <$ MinPts,
        mark O as noise
        return
    else
        select a new cluster_id and mark all objects of $N_\varepsilon(O)$ with this cluster-id and put them into candidate-objects.
        do while candidate-objects is not empty
            select an object from candidate-objects as current_object
            delete current-object from candidate-objects
            retrieve N_ε(current-object)
            if $|N_\varepsilon$(current-object)$| \geq$ MinPts
                select all objects in N_ε(current-object) not yet classified or marked as noise,
                mark all of the objects with cluster_id,
                include the unclassified objects into candidate-objects
        end do
return.

The critical component of the algorithm is the retrieval of its density-reachable neighbourhood. This can be accomplished by an R* tree structure or by an M tree. Thus, DBSCAN relies on the R* tree for speed and scalability in its nearest-neighbour search queries. An illustration of the expand-cluster phase of the algorithm is given below. In Figure 5.3, assume Minpts = 6 and we start with an unclassified object O_1. We find that there are 6 objects in the ε-neighbourhood of O_1. These are: O_1, O_2, O_3, O_4, O_5, and O_6. So, put all these points in the candidate-objects and associate them with cluster-id of O_1 (say, C_1). We shall select an object from the candidate-objects, say O_2. We find that $N_\varepsilon(O_2)$ does not contain an adequate number of points. Hence, we mark this as a noise object. The object O_4 is already marked as a noise (shaded), so there is no further action for this point. Let us select the object O_3 from the candidate-objects. $N_\varepsilon(O_3)$ contains 7 points, O_1, O_3, O_5, O_6, O_9, O_{10} and O_{11}. Among these, O_9, and O_{10} are already marked as noise objects, O_1 and O_3 are already classified, and the others are unclassified. All the seven objects are associated with new cluster-id, C_1. The unclassified objects are included in the candidate-objects for the next iteration.

5.9 BIRCH

BIRCH (**B**alanced **I**terative **R**educing and **C**lustering using **H**ierarchies) is a hierarchical-agglomerative clustering algorithm. It was proposed by Zhang, Ramakrishnan and Livny. It is designed to cluster large datasets of n-dimensional vectors using a limited amount of main memory. BIRCH proposes a special data structure, *CF Tree*, to condense information about subclusters of points. The key idea of the algorithm is as follows.

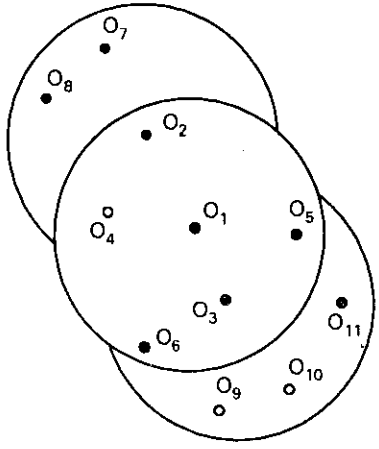


Figure 5.3 Illustration of DBSCAN algorithm. Starting from a candidate object O_1, the cluster expansion process of C_1 is shown here.

BIRCH is based on the principles of agglomerative clustering, that is, at any given stage there are smaller subclusters and the decision at the current stage is to merge subclusters based on some criteria. BIRCH handles this task in a very novel manner. Instead of maintaining all the objects of a subcluster, BIRCH maintains a set of *Cluster Features*(CF) of the sub-cluster. The criteria for merging two subclusters are so defined that decision to merge two subclusters can be taken from the information provided solely by the set of CFs of the respective subclusters. One need not refer to the main data objects for this task. The Cluster Features of different subclusters are maintained in a tree (in a B^+ tree fashion), This tree is called *CF Tree*. The nicety of the algorithm is that it requires just one pass to construct a CF Tree, and the subsequent stages works on this tree rather than the actual database. The last stage, which the proposers of the theory term as optional, requires one more database pass.

Clustering Features and the CF Tree

The concept of clustering features and the CF tree are at the core of BIRCH. A clustering feature is a triple, summarizing the information about a cluster.

Definition 5.8 Cluster Feature Vector

Let C be a cluster containing the data objects $O_1, O_2,\ldots,O_n$. Then, the clustering feature vector (CF) of C is defined as a triple: CF = (n, ls, ss); where n is the number of data objects in C, ls is the linear sum of the data objects, and ss is the square sum of the data objects in C. In other words,

$$|C| = n;$$

$$\sum_{i \in C} O_i = ls$$

$$\sum_{i \in C} O_i^2 = ss$$

Since the objects are multidimensional, the summations and exponentiation in the above expressions are dealt with component-wise. The dimensions of ls and ss are the same as that of the objects in C. For example, if $O_1(x_{11}, x_{12}, x_{13})$, $O_2(x_{21}, x_{22}, x_{23}),\ldots,O_n(x_{n1}, x_{n2}, x_{n3})$, then

$ls = (ls_{(1)}, ls_{(2)}, ls_{(3)})$, such that

$ls_{(1)} = x_{11} + x_{21} + \ldots + x_{n1}$,

$ls_{(2)} = x_{12} + x_{22} + \ldots + x_{n2}$ and

BIRCH assumes that the distance functions between clusters are so defined that they can be expressed in terms of the parameters encoded in the CF vectors. Thus, the algorithm necessarily assumes the database to be numeric in nature. Moreover, it cannot handle all kinds of distance (metric) functions or dissimilarity functions. However, there are many important distance functions that can be described in terms of the CF vectors.

Additive Properties of Cluster Features

Interestingly, the CF vector follows the additive properties. Assume that $CF_1=(n_1, ls_1, ss_1)$ for cluster C_1 and $CF_2 = (n_2, ls_2, ss_2)$ for cluster C_2, such that $C_1 \cap C_2 = \varnothing$. Then, the CF vector of the cluster that is formed by merging the two disjoint clusters is $(n_1+n_2, ls_1+ls_2, ss_1+ss_2)$.

It is easy to see that given the CF vectors of a cluster, the following metric information can be calculated.

Definition 5.9 Centroid

For a cluster $C = \{O_1, O_2,\ldots, O_n\}$, the centroid is defined as an object given by

$$O_{centroid} = \frac{\sum_{i=1}^{n} O_i}{n} = \frac{ls}{n}.$$

Definition 5.10 Radius

The radius R of a cluster $C = \{O_1, O_2,\ldots,O_n\}$, is defined as

$$R = \left[\frac{\sum_{i=1}^{n}(O_i - O_{centroid})^2}{n}\right]^{\frac{1}{2}} = \left[\frac{ss - \frac{2(ls)^2}{n} + \frac{(ls)^2}{n^2}}{n}\right].$$

Definition 5.11 Diameter

The diameter of a cluster $C = \{O_1, O_2, \ldots, O_n\}$, is defined as

$$D = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(O_i - O_j)^2}{n(n-1)}}.$$

Similarly, the following distance parameters between two clusters can also be computed using CF vectors. I leave this exercise to the readers—to express these distance measures in terms of cluster feature vectors.

Definition 5.12 Euclidean Intercluster Distance D_0

For two clusters C_1 and C_2 with cluster centroids $O_{\text{centroid},1}$ and $O_{\text{centroid},2}$, respectively, the *Euclidean intercluster* distance between these clusters is defined as

$D_0(C_1, C_2)$ = The Euclidean distance between $O_{\text{centroid},1}$ and $O_{\text{centroid},2}$.

Thus,

$$D_0(C_1, C_2) = \left[\sum_i (O^i_{\text{centroid},1} - O^i_{\text{centroid},2})^2\right]^{\frac{1}{2}}.$$

where the superscript denotes the components of a vector.

Definition 5.13 Manhattan Intercluster Distance D_1

For two clusters C_1 and C_2 with cluster centroids $O_{\text{centroid},1}$ and $O_{\text{centroid},2}$, respectively, the *Manhattan intercluster* distance between these clusters is defined as

$D_1(C_1, C_2)$ = The Manhattan distance between $O_{\text{centroid},1}$ and $O_{\text{centroid},2}$.

Thus,

$$D_1(C_1, C_2) = \left[\sum_i | O^i_{\text{centroid},1} - O^i_{\text{centroid},2} |\right].$$

Definition 5.14 Average Intercluster Distance D_2

For two clusters C_1 and C_2 with cluster centroids $O_{\text{centroid},1}$ and $O_{\text{centroid},2}$, respectively, the *average intercluster* distance between these clusters is defined as

$$D_2(C_1, C_2) = \left[\frac{1}{n_1 n_2}\sum_{i \in C_1}\sum_{j \in C_2}(O_i - O_j)^2\right].$$

DEFINITION 5.15 AVERAGE INTRACLUSTER DISTANCE

For two clusters C_1 and C_2 with cluster centroids $O_{centroid,1}$ and $O_{centroid,2}$, respectively, the *average intracluster distance* between these clusters is defined as

$$D_3(C_1, C_2) = \left[\frac{1}{(n_1 + n_2)(n_1 + n_2 - 1)} \sum_{i,j \in C_i \cup C_j} (O_i - O_j)^2 \right].$$

We shall now study four phases of the BIRCH clustering process.

PHASE I CONSTRUCTION OF A CF TREE

A CF Tree is a height-balanced tree with two parameters: (i) the branching factor **B**, and (ii) the threshold **T**. Each non-leaf node contains at most **B** entries of the form $[CF_i, child_i]$, where $child_i$ is a pointer to its i^{th} child node and CF_i is the cluster feature of the subcluster represented by this child. So, a non-leaf node represents a cluster made up of all the subclusters represented by its child nodes. A leaf node contains at most **B** entries, each of the form $[CF_i]$. In addition, each leaf node can have two pointers; '*prev*' and '*next*', which can be used to chain all leaf nodes together for an efficient scan. A leaf node represents a cluster made up of all subclusters represented

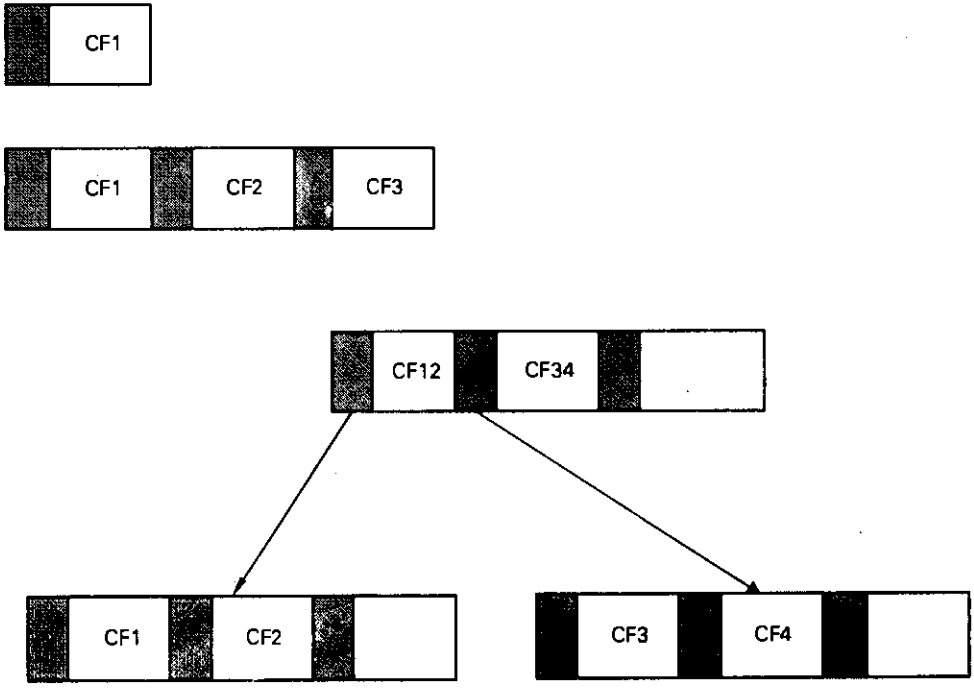


Figure 5.4 CF Tree Construction Process. O_1, O_2, O_3 and O_4 are inserted.

by its entries. All the entries in a leaf node must satisfy the threshold requirements with respect to the threshold value **T**, that is, *the diameter of the subcluster must be less than T.* A CF Tree is built dynamically and incrementally as new data objects are inserted.

It is necessary to understand the insertion process in a CF Tree. It is assumed that the readers are familiar with B Trees and B^+ Trees. Let us assume that a new object O is being inserted into the CF Tree. Let the $CF_0=[1, O, O^2]$. The stages described below are involved in the insertion process.

Identifying the Appropriate Leaf

Starting from the root, O moves recursively downwards by selecting at every step the closest child node according to the chosen distance metric.

Using CF_i and CF_0, the distance between the i^{th} subcluster and the object O is calculated to determine the cluster that is closest to O and, hence, O moves to the corresponding child node.

Modifying the Leaf Node

When the leaf node is reached, it selects the closest leaf entry, say L_i, which is also represented by its CF. At this stage, either O is absorbed into some subcluster in the leaf node or it has a separate entry into this node.

Absorbing O in L_i

L_i represents a subcluster whose diameter is less than the threshold **T**. O is introduced to L_i only if the inclusion O to the subcluster does not violate the threshold condition; that is, the diameter of the new cluster still remains below **T**. Otherwise, O should be introduced as a separate entry into the leaf node.

Introduce O in the Leaf Node

A new entry corresponding to O is added to the leaf and it represents a subcluster with just the single element O.

Since a leaf node can have at most **B** entries, the step of introducing an additional entry to the leaf node is trivial only when there is space on the leaf for this new entry. But in case the leaf node is full, the new entry leads to the splitting of this leaf node.

Splitting of the Leaf Node

The leaf node is split to form two leaf nodes. The node splitting is done by choosing the farthest pair of entries as seeds in the leaf node, and redistributing the remaining entries based on the closest criteria.

Modifying the Path to the Leaf

- When O is absorbed in L_i, the CF vector of L_i is modified with a new value. This, in turn, will affect the cluster features of all the nodes from L_i to the root.

- Similarly, if O_i is added as a separate entry to L_j but without splitting L_j, then the CF vector of the parent node of L_j will be updated and all the other nodes in the path will reflect this change.
- If there is a node split, there is a new entry at the parent node of L_j and this entry consists of the CF vector together with a pointer to the new leaf node. This process may propagate upwards, in case the parent node of L_j does not have enough space to accommodate the new entry. It may be necessary to split the parent as well and so on, up to the root. If the root is split, the height of the tree increases by one.

MERGING REFINEMENT

Suppose that there is a leaf split and the propagation of this split stops at some non-leaf node, N_j. In other words, N_j can accommodate the additional entry resulting from the split. We now scan the node N_j to find the two closest entries. If they are not the pair corresponding to the split, we try to merge them and the corresponding two child nodes. If there are more entries in the two-child node to be accommodated in a merge, we split the merging again by redistributing. During such re-splitting, in case one seed attracts enough merged entries to fill a page, we just put the rest of the entries with the other seed.

EXAMPLE 5.1

The above process is illustrated in Figures 5.4–5.7 for 8 objects $O_1, O_2, \ldots, O_8$ in order. The branching factor **B** is taken to be 3.

For the first three objects O_1, O_2 and O_3, the insertion is trivial, with the CF tree consisting of just one node. When O_4 is inserted, the node is split to two leaf nodes corresponding to the farthest pairs of objects, O_3 and O_4. The other objects are allocated to the leaf nodes based on the closest distance condition. As the second object is closer to O_1, it is in the first leaf node. The root node maintains the details of the two subclusters corresponding to the leaf nodes. When O_5 is introduced, since it is at a distance beyond the threshold **T** from any of the subclusters, one more child node of the root node is created. The details at the root node and the new leaf node are updated. The object O_6 is allocated to the third leaf node according to the closest distance criterion. The objects O_7 and O_8 are allocated to the first leaf node. But since **B**=3, this node is now split into two leaf nodes. The necessary redistribution is illustrated in Figure 5.7.

Thus, Phase I works as follows. It scans the data set record-by-record to form a CF tree. If the tree cannot be accommodated in the main memory, the leaf nodes are reinserted to make the tree smaller. Ideally, with the proper choice of **B** and **T**, the method requires just a one pass scan of the database and it returns a CF Tree.

PHASE II: CONDENSATION OF CF TREE

The size of the CF Tree, generated in Phase I, is dependent on **B** and **T**. A smaller T

Figure 5.5 CF Tree after inserting O_5.

Figure 5.6 CF Tree after O_6, O_7 are inserted. The root node and left-most child node are full

Figure 5.7 CF Tree after O_8 is inserted. Nodes split and the tree expands

generates a larger tree. One can easily see the geometric interpretation of the data objects represented by a CF Tree. At the leaf level, it represents a set of subclusters not exceeding in its diameter by T. The centroids of any two subclusters are separated by the distance T. At most, B number of the closest subclusters at one level are merged to represent a parent node at the next higher level. If T is too small, then it may be that there are too many subclusters at the leaf level. As a result, for large number of points, the resulting CF Tree may be too deep.

Phase II deals with processing the CF Tree by redefining the threshold, so that the tree size becomes manageable. Phase II has yet another aim. It removes the isolated subclusters as outliers during the process of condensing. It scans the leaf entries in the initial CF Tree to rebuild a smaller CF Tree, while removing the outliers and grouping crowded subclusters into larger ones.

Phase III Hierarchical Agglomerative Clustering

Starting with the set of subclusters at the leaf nodes, any one of the conventional hierarchical clustering techniques (global or semi-global) is utilized to get the set of user-specified clusters. The target number of clusters is an input from the user for this phase. This phase employs the clustering technique directly to the subclusters which are represented by their CF vectors. It uses the distance metric, described above, which can be calculated from the CF vectors. One advantage of this method is that the user can either specify the desired number of clusters or the threshold diameter. Phase III also works on the CF Tree and does not require any database scan.

Phase IV Classification

Phase IV uses the centroid of the clusters produced by Phase III as seeds and redistributes the data objects to I, the closest seed, to obtain a set of new clusters. This allows the objects to migrate from one cluster to the other, and it ensures that multiple copies of the same object are included in the same clusters. It may be noted that the CF tree-building algorithm depends on the order of presentation of the data objects, and it is likely that two copies of the same object presented at different instances of the sequence may land up at different nodes.

The CF tree is one of the novel algorithms for data mining. The more efficient version uses a CF* tree as the basic data structure.

5.10 CURE

CURE – Clustering Using Representatives – is yet another interesting algorithm. It is also a hierarchical clustering technique, adopting an agglomerative scheme. CURE

shares certain common motivations with BIRCH. As we have discussed in the previous section, BIRCH maintains CF vectors that are used for merging subclusters during the agglomerative process. The main argument in favour of this strategy is that it avoids the *all-points* (brute-force) method of maintaining all points of the subclusters. One does not have to examine all the data objects of two subclusters to decide whether or not to merge. BIRCH adopts a *centroid point*, in place of *all-points*, paradigm for merging using the distance from the centroid as the merging criterion. We can view all-points and centroid point as two extreme strategies and there can be a host of other strategies that have a trade-off between accuracy and efficiency. Please note that the all-points strategy is an exact measure compared to the approximate measure of the centroid point. Similarly, the all-points strategy is computationally intensive and time consuming, whereas the centroid point is efficient. CURE attempts to strike a balance between these two extremes.

CURE maintains a set of representative points of each subcluster. The representative points are reasonably smaller in number to avoid the inefficiency of the all-points strategy. The representative points are well-scattered within the subcluster so as to properly represent the whole subcluster, rather than the notional single point representative as in the case of centroid. In the event, the subcluster being spherical in shape, the centroid is the best representative of the shape. But most real-life situations cannot guarantee the clusters as being spherical in shape. Maintaining a set of well-scattered representative objects of each subcluster is the single most important aspect of CURE.

The algorithm works as follows. As with any agglomerative algorithm, it begins with every single data object as a cluster and the object itself is the sole representative of the corresponding cluster.

At any given stage of the algorithm, we have a set of subclusters and, associated with each subcluster, we have a set of representative points. The distance between two subclusters is the smallest pair-wise distance between their representative points. For every subcluster, C, its nearest subcluster $C_{nearest}$ is computed at this stage. Define a measure

$$D_{closest}(C) = \text{Distance}(C, C_{nearest}).$$

The subclusters are arranged in the increasing order of $D_{closest}(C)$. Hierarchical clustering works by merging the closest pair of subclusters. The subcluster C corresponding to the smallest value of $D_{closest}(C)$ is the candidate subcluster to be merged with its nearest subcluster $C_{nearest}$ for the next stage. Once the clusters are merged, a new set of representative points are computed for the merged cluster. The merging process continues till the prespecified number of clusters are obtained.

The problem now is to compute the set of representative points of the new subcluster. The set of representative points of individual subclusters may not represent the well-scattered set of points of the merged cluster. The method to recompute the representative

points of the newly-formed cluster is as follows. It begins with the centroid of the new cluster and finds the farthest data object from the centroid. This is the first representative point. In the subsequent iterations, a point in the subcluster is chosen, that is farthest from the previously chosen representative points. The points are then shrunk towards the centroid by a fraction α. The idea of shrinking these points toward the centroid is to get over the problem of the outlier object as a scattered representative.

CURE maintains a heap data structure to determine the closest pair of subclusters at every stage. It also maintains a *k-d* tree for efficient implementation. While implementing the algorithm, one can think of many refinements and adjustments at each stage. For example, one can maintain the pair-wise distances between all the representative points of all subclusters for the efficient computation of the closest pair of subclusters.

CURE is a sampling-based, hierarchical clustering algorithm that is able to discover clusters of arbitrary shapes. However, it relies on vector operations and therefore cannot apparently cluster data in any distance space. The idea of maintaining representative points in this approach is very elegant.

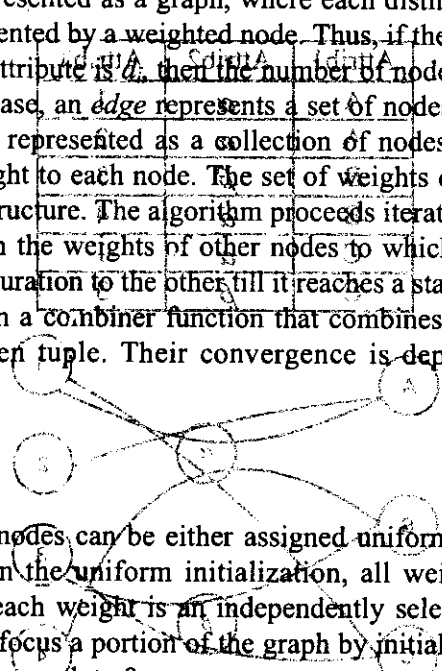
5.11 Categorical Clustering Algorithms

Although algorithms like BIRCH, CURE, CLARANS are suitable for large data sets, these are designed primarily for numeric data. There have been a few proposals for clustering using a categorical data set. The important algorithms are STIRR, ROCK, and CACTUS. One interesting common feature of these three algorithms is that they attempt to model the similarity of categorical attributes in more or less a similar manner. A and B are deemed to be similar if the sets of items with which they co-occur have a large overlap, even though these never co-occur together. ROCK tries to introduce a concept called neighbours and link. Two items have a link if they have a common neighbour. STIRR defines a weighted node and the weights of co-occurring items are propagated and, as a result, two items sharing common co-occurring items will exhibit a similar magnitude of weights. CACTUS also makes use of occurrences as the similarity measure. One can even go one step further to say that if A and B are similar and B and C are deemed similar, then we should be able to infer a weak type of similarity between A and C. In that way, similarity may uncover more distant co-relations. In the following three sections, we shall briefly outline these three algorithms.

5.12 STIRR

STIRR (Sieving Through Iterated Relational Reinforcement), proposed by Gibson, Kleinberg and Raghavan, is an iterative algorithm based on non-linear dynamical systems. The salient features of STIRR are the following.

The database is represented as a graph, where each distinct value in the domain of each attribute is represented by a weighted node. Thus, if there are N attributes and the domain size of the i^{th} attribute is d_i, then the number of nodes in the graph is $\Sigma_i d_i$. For each tuple in the database, an *edge* represents a set of nodes which participate in that tuple. Thus, a tuple is represented as a collection of nodes, one from each attribute type. We assign a weight to each node. The set of weights of all the nodes define the configuration of this structure. The algorithm proceeds iteratively to update the weight of each node, based on the weights of other nodes to which it is connected. Thus, it moves from one configuration to the other till it reaches a stable point. The updating of the weights depends on a combiner function that combines the weights of the nodes participating in a given tuple. Their convergence is dependent on the combiner function.



INITIAL CONFIGURATION

The initial weights of nodes can be either assigned uniformly, or randomly, or by a focussing technique. In the uniform initialization, all weights are set to 1. In the random initialization, each weight is an independently selected random value in the interval [0, 1]. We can focus a portion of the graph by initializing the portion to 1 and the remaining part of the graph to 0.

WEIGHT UPDATE

STIRR iteratively changes the configuration by updating the weight of any single node. The new weight of any node is calculated, based on a *combiner function*. Typically, a combiner function combines the weights of other nodes participating in any tuple with the given node for which the weight is to be updated. We illustrate this step for a specific combiner function in Figure 5.8. A combiner function can simply add all the weights, or can multiply all the weights, or may combine them in some other way.

EXAMPLE 5.2

The concept of weight-update is illustrated in the following diagram. We assume the database given in following table (Table 5.1). The graph corresponding this database is given in Figure 5.8.

Let us assume that at any stage the weight of the node B is w_1 and we are to update the weight of this node.

B participates in two tuples, tuple 3 and tuple 4. For tuple 3, the other nodes are 'α' and 'A'. Thus, the weights w_2 and w_3 are to be combined to get the *combiner*(w_2, w_3) and similarly for the tuple 4, we get this expression as the *combiner*(w_4, w_5). Then, in order to get the weight of B, we add these to the combined weights.

Table 5.1

Attrib1	Attrib2	Attrib3
A	α	1
A	α	2
B	β	3
B	α	4
C	α	1
C	β	3

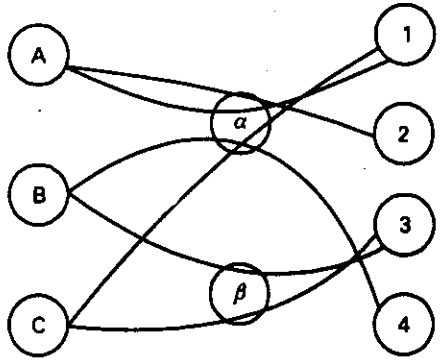


Figure 5.8 Graph for Tuples in Table 5.1

If the combiner function is 'sum', then the new weight of B is

$$w_\alpha + w_4 + w_\beta + w_3.$$

If multiplication is taken as the combiner function, the new weight of B is

$$w_\alpha\, w_4 + w_\beta\, w_3.$$

The algorithm computes the new weights of all the nodes by the weight propagation scheme. Before beginning the next iteration, all the nodes are normalized. There can be two different types of normalization. Global normalization normalizes the weights so that the squared sum of all weights is 1. Local normalization rescales the weights so that the squared sum of all weights for each attribute is 1.

The following table (Table 5.2) gives the change in the weights of each of the nodes for two iterations.

The algorithm proceeds in this manner till it converges to a fixed point. Analyzing the stability is difficult for any arbitrary combiner function. However, for simple combiner functions like sum or multiplication, the system definitely converges to a fixed point. Even if we start from different initial weights, it is likely that the system converges to a fixed configuration. Such fixed-point configurations are called *basins*. It is easy to see that for categorical attributes, the values which are related through

Table 5.2

	A	B	C	α	β	1	2	3	4
Initial wts	1	1	1	1	1	1	1	1	1
Combiners	$\alpha+2+\alpha+1$	$\alpha+4+\beta+3$	$\alpha+1+\beta+3$	A+B+C+1+1+2+4+A	B+3+C+3	A+α+C+α	A+α	B+β+C+β	B+α
Iteration 1 New wts	4	4	4	8	4	4	2	4	2
Normalized wts	0.333	0.338	0.338	0.667	0.338	0.338	0.169	0.338	0.169
Iteration 2 New wts	1.521	1.352	1.521	1.69	1.352	1.69	0.845	1.352	0.845

common tuples influence each other during weight modification. Thus, one does not really require any similarity metric to be defined for categorical attributes.

Interestingly, in order to cluster the set of tuples, STIRR maintains multiple copies of the weights. As a result, the configuration is no longer a one-dimensional vector of weights, but it becomes a matrix of weights. For instance, if we maintain two copies of the graph of 10 nodes, we have a 10 by 2 matrix of weights. The weights of each node are updated independently. But normalization is carried out by converting the matrix to an orthonormal form. This process induces negative weights. However, during the weight modification process, each row of this matrix is treated as a single configuration. The first row is termed as the principal configuration.

Thus, STIRR maintains multiple copies b_1, ..., b_m, called basins, of this set of weighted vertices. The weights on any given vertex may differ across basins; b_1 is called the *principal basin*; b_2, ..., b_m are called *non-principal basins*. Starting with a set of weights on all vertices (in all basins), the system is "iterated" until a fixed-point is reached. When the fixed-point is reached, the weights in one or more of the basins b_2,...,b_m isolate two groups of attribute values on each attribute: the first with large positive weights and the second with large negative weights. The nodes with large positive weights and large negative weights are grouped to determine clusters.

The underlying idea of STIRR is unique, but it may be hard to analyze the stability of the system for any useful combiner functions. It requires rigorous experimentation and fine-tuning of parameters to arrive at a meaningful clustering.

5.13 ROCK

ROCK (**Ro**bust hierarchical-**C**lustering with **K**s) is an adaptation of an agglomerative hierarchical clustering algorithm. An agglomoretive hierarchial clustering normally uses distance-based representatives (such as centroids) to determine similarity between clusters. it is observed that such similarity functions

tend to merge clusters which have disjoint set of attributes. Moreover, it is not possible to extend the concept of centroid to categorical attributes. ROCK makes use of links for defining similarity. Informally, the number of links between two tuples is the number of common neighbours they have in the data set. Starting with each tuple in its own cluster, the two closest clusters are merged until the required number of clusters are obtained. ROCK, instead of working on the whole data set, clusters a sample randomly drawn from the data set, and then partitions the entire data set based on the clusters from the sample. Before stating the algorithm, we shall introduce some definitions.

DEFINITION 5.16 NEIGHBOURS

An object's neighbours are those objects that are considerably similar to it. Let $sim(O_i, O_j)$ be a similarity function that is normalized and captures the closeness between the pair of objects O_i and O_j. The similarity function sim assumes values between 0 and 1. Given a threshold θ (between 0 and 1), a pair of objects O_i and O_j are defined as *neighbours* if

$$sim(O_i, O_j) \geq \theta$$

DEFINITION 5.17 LINKS

The $link(O_i, O_j)$ between the objects is defined as the number of common neighbours between O_i and O_j. Intuitively, if the $link(O_i, O_j)$ is large, then it is more probable that O_i and O_j belong to the same cluster.

ROCK attempts to maximize the sum of $link(O_p, O_r)$ for pairs of objects O_q, O_r belonging to a single cluster and at the same time, to minimize the sum of the $link(O_q, O_r)$ for object pairs belonging to different cluster.

EXAMPLE 5.3

Let us consider the transaction database of Table 4.1. We assume each transaction as one object and define the similarity function as

$$sim(O_i, O_j) = \frac{|O_i \cap O_j|}{|O_i \cup O_j|}.$$

Hence, $sim(O_1, O_2)=1/6$ and $sim(O_2, O_6)=1/2$.

If we set $\theta=1/2$, then we say that the objects O_2 and O_6 are neighbours whereas for the given θ, O_1 and O_2 are not neighbours. Note that $sim(O_{10}, O_{11})=1$, which means that these two objects are identical. On the otherhand, $sim(O_1, O_4)=0$, which means that there is nothing in common between the objects O_1 and O_4.

If we set $\theta=1/6$, then the neighbours of O_1 are O_2, O_3, O_5, O_8, O_9, O_{10}, O_{11} and O_{12}. Similarly, the neighbours of O_6 are O_2, O_3, O_4, O_{10}, O_{11}, and O_{15}.

Thus, O_1 and O_6 have 5 common neighbours. Hence, $link(O_1, O_6)=5$.

DEFINITION 5.18 LINK BETWEEN CLUSTERS

For two clusters, C_i, C_j, define $link[C_i, C_j]$ as follows

$$link[C_i, C_j] = \sum_{O_q \in C_i, O_r \in C_j} link(O_q, O_r)$$

The link-based approach adopts a global perspective of the clustering problem. It captures the global knowledge of neighbouring data points into the relationship between the individual pair of points.

After drawing a random sample from the database, a hierarchical clustering algorithm that employs links is applied to the sample objects. It follows the standard principle of hierarchical clustering. It starts with a singleton object as an individual class and progressively merges the clusters based on the goodness criteria, determined by the link structure. Finally, the clusters involving only the sample objects are used to assign the remaining data objects on the disk to the appropriate clusters.

The goodness measure, $g(C_i, C_j)$, for merging two clusters, C_i, C_j, is defined as

$$g(C_i, C_j) = \frac{link[C_i, C_j]}{\left(n_i + n_j\right)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}}$$

where $link[C_i, C_j]$ stores the number of cross links between clusters C_i and C_j and the denominator expression gives the expected number of cross-links between C_i and C_j. $|C_i| = n_i$ and $n_i^{1+2f(\theta)}$ is expected number of links in C_i; $f(\theta)$ is a function that is dependent on the data set; we define

$$f(\theta) = \frac{1-\theta}{1+\theta}, \quad \theta < 1.$$

The algorithm is described below.

ROCK Algorithm

Input: (S, k) where S is the set of n sampled objects to be clustered and k is the number of desired clusters.

Initially, the link for each pair of objects in S is computed.

for each $s \in S$, $q[s]$ maintains a local heap structure of objects linked to s in decreasing order.

A ***global heap***, Q, is also maintained to indicate the number of clusters that are created so far out of S.

while (size(Q) > k) ***do***
  select a cluster u from Q and select the nearest element v from $q[u]$
  delete v from Q
  w = merge (u, v)
  for each $x \in q[u] \cup q[v]$ ***do***
    link[x, w] = link[x, u] + link[x, v]
    delete u from $q[x]$
    delete v from q [x]

```
            insert w to q[x] and compute g(x, w)
            insert x to q[w] and g(x, w)
            update (Q, x, q[x])
        end
        insert (Q, w, q[w])
    end

    Procedure for computing links
    begin
        compute nbrlist[O_i] for every point i in S
        set link[O_i, O_j] to zero for all i, j
            for i =1 to n do
            N = nbrlist[O_i]
            for j = 1 to |N| – 1 do
                for l = j + 1 to |N| do
                link[N[O_j],N[O_l]] = link[N[O_j], N[O_l]] +1
    end
```

5.14 CACTUS

CACTUS – Clustering Categorical Data Using Summaries – was devised by Ganti, Gehrke and Ramakrishnan. The nicety of CACTUS lies in its problem decomposition. Let us assume that the database D is a set of tuples each having k fields or attributes. The clustering techniques are essentially attempting to cluster the tuples by considering the tuple as an primitive object. CACTUS attempts to split the database vertically and tries to cluster the set of projections of these tuples to only a pair of attributes. The questions that arise are

a. how to find clusters for the tuples only on two attributes, and
b. how to effectively utilize this information for clustering on all the attributes.

Though CACTUS is designed for a categorical database, let us reinforce this concept by giving the following geometric interpretation.

Imagine a set of 3-dimensional vectors. These can be represented as a set of points in the 3-dimensional space. Now consider the projections of these points in 2-dimension, say on the first two dimensions. Consider the clusters formed by these projected points. It may look like Figure 5.9. There are three clusters: C1, C2 and C3. Their projections on the first dimension are S1, S2 and S3. We call these cluster projection sets in Dimension 1, with respect to Dimension 2. We can get another collection of sets Qs for Dimension 1 with respect to Dimension 3. Let us imagine a similar diagram for the Dimension 1 and Dimension 3 of the same set of points. We may get a different set of clusters. Consider their projections on Dimension 1 and let these projections be Q1 and Q2 (assuming that there are just two clusters for this pair of dimensions).

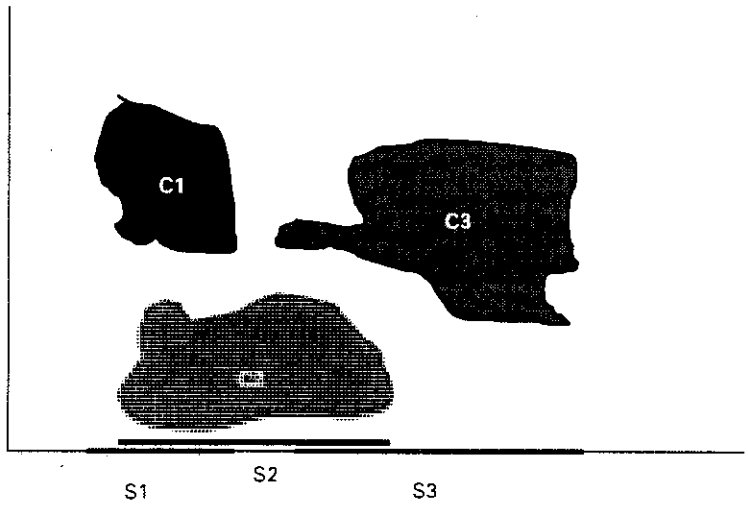


Figure 5.9 Clusters and Their Projections

Knowing S1, S2, S3 and Q1 and Q2, we can derive information about the projections of clusters with all three dimensions on Dimension 1. If we consider intersections of the sets of Ss with the sets of Qs, we get the projections of the clusters when all the three dimensions are taken into account. The main principle is that a cluster on a set of attributes induces a cluster on a subset of the attributes. Thus, if we take the intersecting sets, we get the projections sets of the clusters. CACTUS extends this idea to the categorical attributes. It tries to find clusters for the subsets of attributes, and the clusters for the whole attribute set can be found by intersecting these clusters.

CACTUS has two more important characteristics. The algorithm assumes that categorical domains have a small number of attribute values (i.e., the domain sizes are small). The central idea of CACTUS is that the summary information constructed from the data set is sufficient for discovering well-defined clusters. The properties that the summary information typically fits into main memory and that it can be efficiently constructed in a single scan of the data set are the key ideas of the algorithm.

Based on the foregoing discussion, we shall now describe the algorithm in detail. However, in the beginning let us define some terms to facilitate the understanding of new concepts and to visualize the clusters in terms of individual attributes.

Let us consider two attribute values of two different attributes in the database. Say, a_i of attribute type A and a_j of attribute type B. There may be tuples where a_i and a_j co-occur. The support of these two values in the database is the proportion of tuples in which they appear together. If this support exceeds a prespecified value, we say that these values are strongly connected. D_A denotes domain of the attribute A; s denotes the support value.

DEFINITION 5.19 STRONGLY CONNECTED ATTRIBUTE VALUES

Let $a_i \in D_A$, $a_j \in D_B$, and $\alpha > 1$. The attribute values a_i and a_j are *strongly connected* with respect to domain D if

$$s_D(a_i, a_j) > \frac{\alpha |D|}{|D_A||D_B|}.$$

The function $\sigma^*_D(a_i, a_j)$ is defined as

$$\sigma^*_D(a_i, a_j) = \begin{cases} s_D(a_i, a_j), & \text{if } a_i \text{ and } a_j \text{ are strongly connected.} \\ 0, & \text{otherwise} \end{cases}$$

The above definition can be extended to define strongly connected value sets. Let a_i be an attribute value and S a set of attribute values of another attribute type. We say that a_i is strongly connected to S if it is strongly connected to each element of S. Similarly, two attribute value sets S and R are said to be strongly connected if each value a in S is strongly connected to R, and each value b in R is strongly connected to S. We use this concept to define a cluster.

DEFINITION 5.20 CLUSTER

Let C_i be the attribute value set of i^{th} attribute; a subset of the domain D_i, such that $|C_i| > 1$. Then, $C = <C_1, \ldots, C_n>$ is said to be a *cluster* over the database of n attributes if the following three conditions are satisfied:

1. For all $i, j \in \{1, \ldots, n\}$, $i \neq j$, C_i and C_j are strongly connected.
2. For all $i, j \in \{1, \ldots, n\}$, $i \neq$ j, there exists no $C_i' \supset C_i$ such that for all $j \neq i$, C_i' and C_j are strongly connected.
3. $$s_D(C) \geq \frac{\alpha |D| \times |C_1| \times \cdots \times |C_n|}{|D_1| \times |D_2| \times \cdots \times |D_n|}$$

DEFINITION 5.21 *K*-CLUSTER

A cluster on k attributes is called a k-cluster. It is also known as a subspace cluster.

DEFINITION 5.22 CLUSTER PROJECTION

A cluster on a database that is a projection of the original database D is called a cluster projection, with respect to the attributes present in the projection operation. C_i above is called the cluster projection of C on attribute A_1.

DEFINITION 5.23 SIMILARITY MEASURE OF ELEMENTS OF ONE ATTRIBUTE

We define a similarity measure between two values of an attribute, i.e., for $a_1, a_2 \in D_i$

with respect to another attribute A_j, as the number of elements of D_j that are strongly connected to both a_1, and a_2.

$$\Upsilon^j(a_1, a_2) = \left|\{x \in D_j : \sigma_D^*(a_1, x) > 0 \text{ and } \sigma_D^*(a_2, x) > 0\}\right|$$

DEFINITION 5.24 INTER-ATTRIBUTE SUMMARY

Let $A_1, \ldots, A_n$ be a set of categorical attributes with domains $D_1, \ldots, D_n$, respectively, and let D be a data set. The inter-attribute summary $\sum_{IJ}$ is defined as

$$\sum_{IJ} = \{\sum_{ij} \mid i, j \in \{1, \ldots, n\}, \text{and } i \neq j\}$$

where $\sum_{ij} = \{(a_i, a_j, \sigma_D^*(a_i, a_j)) \mid a_i \in D_i, a_j \in D_j \text{ and } \sigma_D^*(a_i, a_j) > 0\}$.

DEFINITION 5.25 INTRA-ATTRIBUTE SUMMARY

The intra-attribute summary $\sum_{II}$ is defined as

$$\sum_{II} = \{\sum_{ii}^j \mid i, j \in \{1, \ldots, n\}, \text{and } i \neq j\}\}$$

where $\sum_{ij}^j = \{(a_i, a_j, \Upsilon^j(a_{i1}, a_{i2})) \mid a_{i1}, a_{i2} \in D_i \text{ and } \Upsilon^j(a_{i1}, a_{i2}) > 0\}$.

CACTUS first identifies the cluster projections on an attribute for the 2 clusters involving this attribute. Then, it generates an intersecting set to represent the cluster projection on this attribute for n-cluster (involving all the attributes). Once all the cluster projections on individual attributes are generated, these are synthesized to get the clusters of the database. Thus, the major steps of CACTUS are

1. Finding cluster projections $CP_i[i, j]$ on a given attribute A_i with respect to another attribute A_j in a 2-cluster involving these two attributes. This is done for all attributes A_i.
2. Intersecting these $CP_i[i, j]$ for a fixed A_i to get $CP_i[1, 2, \ldots, k]$; that is, a cluster projection of A_i with respect to all the attributes. It is also done for all A_i.
3. Synthesizing the set of $CP_i[1, 2, \ldots, k]$, for all i, to get the clusters.

We shall discuss each step separately. The most interesting aspect of these steps are that they can be computed using the inter-attribute and intra-attribute summary. It is not necessary to refer to the original database. Hence, we assume that these summaries are available as the outcome of preprocessing. Needless to say, these summaries can be computed by the level-wise algorithm discussed in Chapter 4.

CLUSTERING PHASE

The clustering phase is a two-step process. In the first step, each attribute's cluster projections are computed. In the second step, candidate clusters on sets of attributes

from the cluster projections are computed. This method contains a set of modules. It first finds the set of distinguishing sets, then extends these distinguishing sets to determine the cluster projection of 2-clusters. Let us first understand the concept of distinguishing set.

DEFINITION 5.26 DISTINGUISHING SET

A distinguishing set DS, on an attribute A_i, is a set of values having the following properties:

- All pairs of values in DS are strongly connected (refer to the similarity measure above).

To find a distinguishing set, the algorithm uses the principle of downward closure property (Chapter 4). It first finds the pair of values which are strongly connected. It then extends these pairs in a level-wise fashion to get k-sets of strongly connected elements, every time deleting the superset whose at least one proper subset violates the desired property.

Distinguishing sets are extended further. Let $DS_i[j]$ be a distinguishing set of A_i with respect to A_j. Let us identify the values in A_j which are strongly connected to the values in $DS_i[j]$. For these attribute values in A_j, we refer to the i^{th} attribute to determine other values of A_i which are strongly connected and add these values to $DS_i[j]$.

INTERSECTION OF CLUSTER PROJECTIONS

Once we have computed the collection of all the distinguishing sets $DS_i[j]$ and their extensions, we intersect these collections over all j to determine the cluster projections $CP_i[1, 2, \ldots, k]$ for the attribute A_i. Thus, once the cluster projections on A_i of 2-clusters over attribute pairs (A_i, A_j) are found using the intersection operation on the sets, the set of cluster projections on A_i of clusters over $\{A_1, A_2, \ldots, A_n\}$ can be found. Thus, the clusters so obtained are the candidate clusters. This can be carried out as a main memory operation.

We employ a level-wise principle for the synthesis of cluster projection. We start with cluster-projections on A_1 and then, extend them to clusters over (A_1, A_2), then to clusters over (A_1, A_2, A_3), and so on. Let C_i be the set of cluster projections on attribute A_i. Let C^k denote the set of candidate clusters over $A_1, \ldots, A_k$. Hence $C^1=C_1$. We successively generate C^{j+1} from C^j by checking the summarized information.

VALIDATION PHASE

This step scans through the database to compute the support for each of the candidate clusters.

CACTUS requires only two scans of the database and hence, it is fast and stable.

The algorithm can find clusters in subsets of all attributes also, and thus can perform subspace clustering.

5.15 Conclusion

We have studied quite a few algorithms, each advocating a new idea.

Let us attempt to extract the general principles. The first one is sampling the database. When the database is very large, randomly sampling the data for clustering has proved to be an efficient approach. We have seen that CLARA, ROCK, and CURE follow this approach. Another important feature is summarization. It is worthwhile to extract some features from the database so that these extracted features carry sufficient information for clustering and hence, one does not have to refer to the original database. BIRCH, CACTUS, and CURE follow this approach. For categorical data, however, co-occurrence strength is uniformly taken as the similarity measure. One can perhaps think of some other measure. Another important aspect of the algorithms is the vertical partitioning of the database by projecting on a subset of attributes. CACTUS adopts this for categorical attributes.

Further Reading

Clustering is another well-studied subject and, indeed, a very vast one. *k*-means algorithms can be found in Pattern Recognition and Image Processing literature. One of the very recent *k*-means algorithm is given in [Alsabti, 1998]. A general study on clustering is very well dealt with in [Jain, 1988] and in [Anderberg, 1973]. PAM was proposed by Kaufman *et al.* [Kaufman, 1990]. CLARANS was introduced by Ng in [Ng, 1994]. DBSCAN was proposed by Ester *et al.* [Ester, 1995]. BIRCH was proposed by Zhang *et al.* [Zhang, 1996]. Guha *et al.* introduced the CURE algorithm in [Guha, 1998] and the ROCK algorithm [Guha, 1999]. CACTUS was designed by Ramakrishnan and his team [Ganti, 1999].

There are many other important algorithms like BUBBLE, MAFIA [Goil, 2000], WaveCluster [Sheikholeslami, 1998], ITERATES [Biswas, 1995], CHAMELON [Karyapis, 1997] etc. Wavecluster is for numeric, mutlidimensional data and it uses wavelet transforms for the purpose of clustering. BUBBLE is designed for arbitrary metric space and draws its foundation from BIRCH.

Exercises

1. What is clustering? What are the different clustering techniques?
2. Discuss the importance of similarity metric in clustering. Why it is difficult to handle categorical data for clustering?

3. Describe the working of the PAM algorithm. Compare its performance with CLARA and CLARANS.
4. How is CLARANS different from CLARA? Illustrate this using a small example.
5. Describe the working of the DBSCAN algorithm. Explain the concept of a cluster as used in DBSCAN.
6. Explain the concept of a cluster as used in ROCK. Compare this definition of cluster with that of the DBSCAN.
7. Define a core object. Define density reachability. Why is density reachability not a symmetric function? What is a noise object?
8. BIRCH is independent of any clustering algorithm. It is just a clustering paradigm. Comment.
9. How are CF Tree operations different from B^+ Tree operations. Describe the operations for CF Tree.
10. Discuss the importance of cluster features. How do these help in clustering large databases. How are these different from the cluster representatives used in CURE?
11. What are the different phases of BIRCH? How are they important in clustering?
12. CURE is a sampling-based hierarchical clustering. Justify.
13. Describe the features of ROCK that are necessary for categorical attributes.
14. Define the following
    a. k-clusters
    b. strongly connected attributes
    c. inter-attribute summary
    d. intra-attribute summary
    e. cluster projection
15. Describe the underlying principle of the CACTUS algorithm.
16. List the different features that can be computed from the three cluster features. Which of them are additive function of CFs?
17. Describe the salient features of CURE clustering techniques.
18. For STIRR, complete the example given in the text to reach the fixed point.
19. In STIRR, suppose that the combiner function is the max function. Compute the fixed-point for the example in the text.
20. PAM requires prior specification of the number of clusters. True or false?
21. DBSCAN is a hierarchical divisive algorithm. True or false?
22. If C is a cluster in n-dimension, its projection in n–1 dimension may be a single cluster or more than one cluster. True or false?
23. If the medoid O_j and unselected object O_h swap roles, it is acceptable if the cost of swapping is negative. True or False?
24. If noise object is one that is not density reachable from any other object. True or false?
25. In a cluster as defined in DBSCAN, any two objects are density connected. True or false?

26. Express the average intra-cluster distance in terms of the cluster features.
27. Distinguish between global and local normalization in STIRR.
28. Define a link and a neighbour as defined in ROCK.
29. What are the similarities among the three algorithms: ROCK, STIRR and CACTUS?

Bibliography

Agarwal Pankaj K., and Procopiuc Cecilia M. Exact and approximation algorithms for clustering. In *Proceedings of SODA*, 1998. (extended abstract)

Aggarwal C., Procopiuc C., Wolf J.L., Yu P.S., and Park J.S. A framework for finding projected clusters in high dimensional spaces. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1999.

Agrawal R., Gehrke J., Gunopulos D., and Raghavan P. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1998.

Alsabti K., Ranka S., and Singh V. An efficient *k*-Means clustering algorithm. In *IPPS/SPDP Workshop on High Performance Data Mining*, Orlando, Florida, 1998.

Anderberg M.R. *Cluster Analysis for Applications*. Academic Press, NY, 1973.

Ankerst M., Breunig M., H.-P. Kriegel, and Sander J. OPTICS: Ordering points to identify the clustering structure. In *Proceedings ACM SIGMOD International Conference on Management of Data*, Philadelphia, PA, 1999.

Biswas G., Weinberg J., and Li C. ITERATE: A conceptual clustering method for knowledge discovery in databases. *Artificial Intelligence in the Petroleum Industry*, Paris, France, 1995, pp. 111–139.

Bradley P.S., Mangasarian O.L., and Street W.N. Clustering via concave minimization. *Advances in Neural Information Processing Systems* **9**, MIT Press, Cambridge, MA 1997, 368–374.

Bradley P.S., and Mangasarian O.L. K-plane clustering. *Mathematical Programming Technical Report 98-08*, Department of Computer Science, University of Wisconsin—Madison, August 1998.

Brinkhoff T., and H.-P. Kriegel. The impact of global clustering on spatial database systems. In *Proceedings of 20th International Conference on Very Large Databases*, Santiago, Chile, 1994, pp. 168–179.

Castro V.E. Convex group clustering of large geo-referenced data sets. In *11th Canadian Conference on Computational Geometry*, University of British Columbia, Vancouver, Canada, 1999.

Charikar M., Chekuri C., Feder T., and Motwani R. Incremental clustering and dynamic information retrieval. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 1997, pp. 626–635.

Eldershaw C., and Hegland M. Cluster analysis using triangulation. In *Computational Techniques and Applications*, 201–208, World Scientific, Singapore, 1997.

Ester M., H.-P. Kriegel, and Xu X. A database interface for clustering in large spatial databases. In *Proceedings of 1st International Conference on Knowledge Discovery & Data Mining*, Montreal, Canada, 1995, pp. 94–99.

Ester M., H.-P. Kriegel, Sander J., and Xu X. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings 2nd International Conference on Knowledge Discovery and Data Mining* (KDD-96), Portland, OR, 1996, pp. 226–231.

Ester M., H.-P. Kriegel, Sander J., and Xu X. Clustering for mining in large spatial databases. Special Issue on Data Mining, *KI-Journal*, ScienTec Publishing, No. **1**, 1998.

Ester M., H.-P. Kriegel, Sander J., Wimmer M., and Xu X. Incremental clustering for mining in a data warehousing environment. In *Proceedings 24th International Conference on Very Large Databases*, New York, 1998, pp. 323–333.

Fisher D. Iterative optimization and simplification of hierarchical clusterings. *Technical Report CS-95-01*, Department of Computer Science, Vanderbilt University, 1995.

Ganti V., Ramakrishnan R., Gehrke J., Powell A. and French J. Clustering large data sets in arbitrary metric spaces. In *Proceedings of the International Conference on Data Engineering*, 1999.

Ganti V., Gehrke J., and Ramakrishnan R. CACTUS: Clustering categorical data using summaries. In *Proceedings of ACM SIGKDD, International Conference on Knowledge Discovery & Data Mining*, San Diego, CA, USA, 1999.

Gibson D., Kleinberg J., and Raghavan P. Clustering categorical data: an approach based on dynamical systems. In *Proceedings of the 24th VLDB Conference*, New York, USA, 1998.

Goil S., Harsha and Choudhary Alok. MAFIA: Efficient and scalable subspace clustering for very large data sets. Submitted for publication to ICDE, 2000.

Gonzalez T.F. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, **38**, 1985, pp. 293–306.

Guha S., Rastogi R., and Shim K.. CURE: An efficient algorithm for clustering large databases. In *Proceedings of ACM-SIGMOD 1998 International Conference on Management of Data*, Seattle, 1998.

Guha S., Rastogi R., and Shim K. ROCK: A robust clustering algorithm for categorical attributes. In *Proceedings of the IEEE International Conference on Data Engineering*, Sydney, March 1999.

Han Eui-Hong (Sam), Karypis George, Kumar Vipin and Mobasher B.. Clustering based on association rule hypergraphs. In *SIGMOD'97 Workshop on Research Issues on Data Mining and Knowledge Discovery*, 1997.

Han E.H., Karyapis G., Kumar V., and Mobasher B. Clustering in a high-dimensional space using hypergraph models. *Technical Report 97-019*, Department of Computer Science, University of Minnesota.

Han Eui-Hong, Karyapis George, Kumar Vipin and Mobasher B. Hypergraph based clustering in high-dimensional data sets: a summary of results. *Bulletin of the Technical Committee on Data Engineering*, **21**, No. 1, March 1998.

Hinneburg A., and Keim D. Cluster discovery methods for large databases: From the past to the future, tutorial session. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1999.(paper)

Huang Z. A fast clustering algorithm to cluster very large categorical data sets in data mining. In *Proceedings of SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, 1997.

Jain A.K., and Dubes R.C. *Algorithms for Clustering Data*. Prentice-Hall, NJ, 1988.

Karyapis G., Han E.H., and Kumar V. CHAMELEON: A hierarchical clustering algorithm using dynamic modelling. In *IEEE Computer, Special Issue on Data Analysis and Mining*.

Li C., and Biswas G. Conceptual clustering with numeric and nominal mixed data – A new similarity based system. *IEEE Transactions on Knowledge and Data Engineering*, in review, July 1996.

Ng R.T, and Han J. Efficient and effective clustering methods for spatial data mining. In *Proceedings of 1994 International Conference on Very Large DataBases (VLDB'94)*, Santiago, Chile, September 1994, pp. 144–155.

Ohta Y., Okahama S., and Yugami N. *Clustering Algorithm for Large-scale Databases*. Fujitsu Laboratories Ltd.

Procopiuc Cecilia M. *Clustering Problems and their Applications: A Survey.* Department of Computer Science, Duke University.

Ramkumar G.D., and Swami A. Clustering data without distance functions. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 1998.

Sheikholeslami G., Chatterjee S., and Zhang A. WaveCluster: A multi-resolution clustering approach for very large spatial databases. In *24th International Conference on Very Large Databases*, New York City, August 24–27, 1998.

Xu X., Ester M., H.-P. Kriegel, and Sander J., A distribution-based clustering algorithm for mining in large spatial databases. In *Proceedings 14th International Conference on Data Engineering (ICDE'98)*, Orlando, FL, 1998, pp. 209–218.

Yu D., Chatterjee S., and Sheikholeslami G.. Efficiently detecting arbitrary shaped clusters in very large data sets with high dimensions. Department of Computer Science and Engineering, SUNY Buffalo, *Technical Report*, November 1, 1998.

Zhang Tian, Ramakrishnan Raghu, and Livny Miron. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1996.

6

DECISION TREES

6.1 INTRODUCTION

The classification of large data sets is an important problem in data mining. The classification problem can be simply stated as follows. For a database with a number of records and for a set of classes such that each record belongs to one of the given classes, the problem of classification is to decide the class to which a given record

belongs. But there is much more to this than just simply classifying. The classification problem is also concerned with generating a description or a model for each class from the given data set. Here, we are concerned with a type of classification called supervised classification. In supervised classification, we have a training data set of records and for each record of this set, the respective class to which it belongs is also known. Using the training set, the classification process attempts to generate the descriptions of the classes, and these descriptions help to classify the unknown records. In addition to the training set, we can also have a test data set which is used to determine the effectiveness of a classification. There are several approaches to supervised classifications. *Decision trees* (essentially, *Classification trees*) are especially attractive in the data-mining environment as they represent rules. Rules can readily be expressed in natural language and are easily comprehensible. Rules can also be easily mapped to a database access language, like SQL.

This chapter is concerned with decision trees as techniques for data mining. Though the decision-tree method is a well-known technique in statistics and machine learning, these algorithms are not suitable for data mining purposes. The specific requirements that should be taken into consideration while designing any decision tree construction algorithms for data mining are that

a. the method should be efficient in order to handle a very large-sized database, and
b. the method should be able to handle categorical attributes.

The recent developments in data mining research have led to some interesting decision tree algorithms. This chapter discusses most of the major algorithms on this topic. We shall begin with studying, in Section 6.2, the basic features of decision tree building. Section 6.3 discusses the generic algorithm of decision tree construction. Almost all the known algorithms follow the principles outlined in this section. We shall also outline three major design considerations that should be taken into account while studying a specific algorithm. We shall also study two important measures in Section 6.5, namely Entropy-based gain and the Gini Index. Section 6.6 deals with the method of determination of splitting points for both numerical and categorical attributes.

The specific algorithms are discussed from Section 6.8 onwards. In sections 6.8–6.10, we shall discuss three important and well-known algorithms, which provide the foundation for all subsequent algorithms. In fact, all other algorithms are only adaptions of these algorithms to suit different aspects of data mining. In Section 6.13, we shall discuss one of the recent and popular algorithms, namely SPRINT. In this section we also study SLIQ, which is the initial version of SPRINT. These algorithms demonstrate a process that avoids repeated sorting—one of major computational overheads in earlier algorithms. Historically, SLIQ was proposed first and SPRINT is an improvement on SLIQ. Both are based on a similar principle. The next section outlines the specific improvement that SPRINT exhibits over SLIQ.

Besides the exact methods, in order to efficiently handle large data set, there are also approximate methods of construction of decision trees; we shall discuss these

methods in Section 6.14. There are generally two approaches for devising approximate construction. They are sampling and discretization. CLOUDS combines both these approaches. It carries out a kind of discretization on a sample of the original training set. The construction process proceeds on this discretized version of the sample. But in order to get an accurate construction from a coarse construction, the algorithm employs a hill-climbing heuristic to estimate the gini indices of the coarse intervals. It exploits the features of the gini function and reduces the search space by eliminating certain intervals.

In Section 6.15, we shall discuss yet another approximate algorithm, with the acronym BOAT. BOAT adopts a bootstraping technique to determine a coarse decision tree. However, it subsequently searches only one confidence interval per attribute at each node.

In Section 6.18 , we shall study the well-known pruning strategy—MDL pruning strategy. The pruning process is generally a post-processing step and is employed after the complete construction of the decision tree.

A question that comes to mind is whether the pruning can be integrated with the tree construction process. We have noted that CHAID follows this principle. However, SPRINT employs a MDL pruning step after the full decision tree is expanded. PUBLIC is one of the recent algorithms which attempts to integrate these two steps. Section 6.19 focusses on this method of decision tree building.

6.2 What is a Decision Tree?

A decision tree is a classification scheme which generates a tree and a set of rules, representing the model of different classes, from a given data set. The set of records available for developing classification methods is generally divided into two disjoint subsets—a *training set* and a *test set*. The former is used for deriving the classifier, while the latter is used to measure the accuracy of the classifier. The accuracy of the classifier is determined by the percentage of the test examples that are correctly classified.

We categorize the attributes of the records into two different types. Attributes whose domain is numerical are called the *numerical attributes*, and the attributes whose domain is not numerical are called the *categorical attributes*. There is one distinguished attribute called the *class label*. The goal of the classification is to build a concise model that can be used to predict the class of the records whose class label is not known.

Example 6.1

In order to have a clear idea of a decision tree, let us consider the following data sets—the training data set (see Table 6.1) and the test data set (see Table 6.2). The data

Table 6.1 Training Data Set

OUTLOOK	TEMP(F)	HUMIDITY(%)	WINDY	CLASS
sunny	79	90	true	no play
sunny	56	70	false	play
sunny	79	75	true	play
sunny	60	90	true	no play
overcast	88	88	false	no play
overcast	63	75	true	play
overcast	88	95	false	play
rain	78	60	false	play
rain	66	70	false	no play
rain	68	60	true	no play

set has five attributes. There is a special attribute: the attribute *class* is the class label. The attributes, *temp* (temperature) and *humidity* are numerical attributes and the other attributes are categorical, that is, they cannot be ordered. Based on the training data set, we want to find a set of rules to know what values of *outlook*, *temperature*, *humidity* and *wind*, determine whether or not to play golf. Figure 6.1 gives a sample decision tree for illustration.

In the above tree (Figure 6.1), we have five leaf nodes. In a decision tree, each leaf node represents a rule. We have the following rules corresponding to the tree given in Figure 6.1.

RULE 1 *If it is sunny and the humidity is not above 75%, then play.*
RULE 2 *If it is sunny and the humidity is above 75%, then do not play.*
RULE 3 *If it is overcast, then play.*
RULE 4 *If it is rainy and not windy, then play.*
RULE 5 *If it is rainy and windy, then don't play.*

Please note that this may not be the best set of rules that can be derived from the given set of training data.

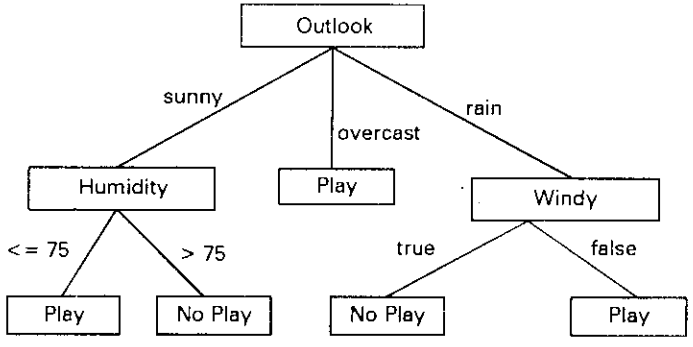


Figure 6.1 A Decision Tree

The classification of an unknown input vector is done by traversing the tree from the root node to a leaf node. A record enters the tree at the root node. At the root, a test is applied to determine which child node the record will encounter next. This process is repeated until the record arrives at a leaf node. All the records that end up at a given leaf of the tree are classified in the same way. There is a unique path from the root to each leaf. The path is a rule which is used to classify the records.

In the above tree, we can carry out the classification for an unknown record as follows. Let us assume, for the record, that we know the values of the first four attributes (but we do not know the value of *class* attribute) as

outlook= rain; temp = 70; humidity = 65; and windy= true.

We start from the root node to check the value of the attribute associated at the root node. This attribute is the *splitting attribute* at this node. Please note that for a decision tree, at every node there is an attribute associated with the node called the *splitting attribute*. In our example, *outlook* is the splitting attribute at root. Since for the given record, *outlook = rain*, we move to the right-most child node of the root. At this node, the splitting attribute is *windy* and we find that for the record we want classify, *windy = true*. Hence, we move to the left child node to conclude that the class label is "no play".

Note that every path from root node to leaf nodes represents a rule. It may be noted that many different leaves of the tree may refer to the same class labels, but each leaf refers to a different rule.

The accuracy of the classifier is determined by the percentage of the test data set that is correctly classified. Consider the following test data set (Table 6.2).

Table 6.2 Test Data Set

OUTLOOK	TEMP(F)	HUMIDITY(%)	WINDY	CLASS
sunny	79	90	true	play
sunny	56	70	false	play
sunny	79	75	true	no play
sunny	60	90	true	no play
overcast	88	88	false	no play
overcast	63	75	true	play
overcast	88	95	false	play
rain	78	60	false	play
rain	66	70	false	no play
rain	68	60	true	play

We can see that for Rule 1 there are two records of the test data set satisfying *outlook= sunny* and *humidity* ≤ 75, and only one of these is correctly classified as *play*. Thus, the accuracy of this rule is 0.5 (or 50%). Similarly, the accuracy of Rule 2 is also 0.5 (or 50%). The accuracy of Rule 3 is 0.66.

EXAMPLE 6.2

At this stage, let us consider another example to illustrate the concept of categorical attributes. Consider the following training data set (Table 6.3). There are three attributes, namely, *age, pincode* and *class*. The attribute *class* is used for class label.

Table 6.3 Another Example

ID	AGE	PINCODE	CLASS
1	30	5600046	C1
2	25	5600046	C1
3	21	5600023	C2
4	43	5600046	C1
5	18	5600023	C2
6	33	5600023	C1
7	29	5600023	C1
8	55	5600046	C2
9	48	5600046	C1

The attribute *age* is a numeric attribute, whereas *pincode* is a categorical one. Though the domain of pincode is numeric, no ordering can be defined among pincode values. You cannot derive any useful information if one pin-code is greater than another pincode. Figure 6.2 gives a decision tree for this training data. The splitting attribute at the root is *pincode* and the splitting criterion here is *pincode = 500 046*. Similarly, for the left child node, the splitting criterion is *age* $\leq$ *48* (the splitting attribute is *age*). Although the right child node has the same attribute as the splitting attribute, the splitting criterion is different.

Most decision tree building algorithms begin by trying to find the test which does the best job of splitting the records among the desired categories. At each succeeding level of the tree, the subsets created by the preceding split are themselves split according to whatever rule works best for them. The tree continues to grow until it is no longer possible to find better ways to split up incoming records, or when all the

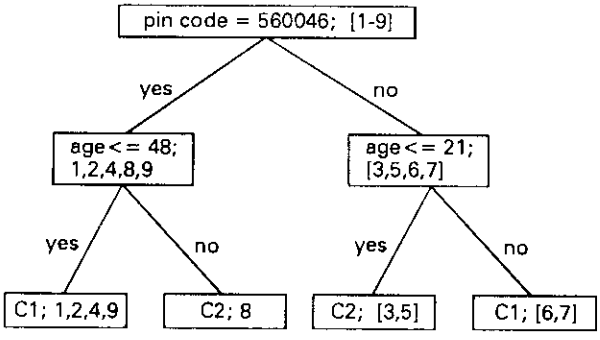


Figure 6.2 A Decision Tree

records are in one class. In Figure 6.2, we see that at the root level we have 9 records. The associated *splitting criterion* is *pincode = 500 046.* As a result, we split the records into two subsets, Records 1, 2, 4, 8 and 9 are to the left child node and the remaining to the right node. This process is repeated at every node.

A decision tree construction process is concerned with identifying the splitting attributes and splitting criteria at every level of the tree. There are several alternatives and the main aim of the decision tree construction process is to generate simple, comprehensible rules with high accuracy.

Some rules are apparently better than others. In the above example, we see that Rule 3 is simpler than Rule 1. The measure of simplicity is the number of antecedents of the rule. It may happen that another decision tree may yield a rule like: "if the temperature lies between 70° F and 80° F, and the humidity is between 75% and 90%, and it is not windy, and it is sunny, then play". Naturally, we would prefer a rule like Rule 1 to this rule. That is why simplicity is sought after.

Sometimes the classification efficiency of the tree can be improved by revising the tree through some processes like pruning and grafting. These processes are activated after the decision tree is built.

Advantages and Shortcomings of Decision Tree Classifications

The major strengths of the decision tree methods are the following:

- decision trees are able to generate understandable rules,
- they are able to handle both numerical and the categorical attributes, and
- they provide a clear indication of which fields are most important for prediction or classification.

Some of the weaknesses of the decision trees are:

- some decision trees can only deal with binary-valued target classes. Others are able to assign records to an arbitrary number of classes, but are error-prone when the number of training examples per class gets small. This can happen rather quickly in a tree with many levels and/or many branches per node.
- the process of growing a decision tree is computationally expensive. At each node, each candidate splitting field is examined before its best split can be found.

6.3 Tree Construction Principle

After having understood the basic features of decision trees, we shall now focus on the methods of building such trees from a given training data set. Based on the foregoing discussion, we shall formally define few concepts for our study.

DEFINITION 6.1 SPLITTING ATTRIBUTE

With every node of the decision tree, there is an associated attribute whose values determine the partitioning of the data set when the node is expanded.

DEFINITION 6.2 SPLITTING CRITERION

The qualifying condition on the splitting attribute for data set splitting at a node, is called the splitting criterion at that node. For a numeric attribute, the criterion can be an equation or an inequality. For a categorical attribute, it is a membership condition on a subset of values.

All the decision tree construction methods are based on the principle of recursively partitioning the data set till homogeneity is achieved. We shall study this common principle later and discuss in detail the features of different algorithms individually.

The construction of the decision tree involves the following three main phases.

- *Construction phase* The initial decision tree is constructed in this phase, based on the entire training data set. It requires recursively partitioning the training set into two, or more, sub-partitions using a *splitting criteria,* until a stopping criteria is met.
- *Pruning phase* The tree constructed in the previous phase may not result in the best possible set of rules due to over-fitting (explained below). The pruning phase removes some of the lower branches and nodes to improve its performance.
- *Processing the pruned tree* to improve understandability.

Though these three phases are common to most of the well-known algorithms, some of them attempt to integrate the first two phases into a single process.

THE GENERIC ALGORITHM

Most of the existing algorithms of the construction phase use *Hunt's method* as the basic principle in this phase. Let the training data set be T with class-labels $\{C_1, C_2, \ldots, C_k\}$. The tree is built by repeatedly partitioning the training data, using some criterion like the goodness of the split. The process is continued till all the records in a partition belong to the same class.

- **T is homogeneous** T contains cases all belonging to a single class C_j. The decision tree for T is a leaf identifying class C_j.
- **T is not homogeneous** T contains cases that belong to a mixture of classes. A test is chosen, based on a single attribute, that has one or more mutually exclusive outcomes $\{O_1, O_2, \ldots, O_n\}$. T is partitioned into the subsets $T_1, T_2, T_3, \ldots, T_n$, where T_i contains all those cases in T that have the outcome O_i of the chosen test. The decision tree for T consists of a decision node identifying the test, and one branch

for each possible outcome. The same tree building method is applied recursively to each subset of training cases. Most often, n is chosen to be 2 and hence, the algorithm generates a binary decision tree.

- **T is trivial** T contains no cases. The decision tree T is a leaf, but the class to be associated with the leaf must be determined from information other than T.

The generic algorithm of decision tree construction outlines the common principle of all algorithms. Nevertheless, the following aspects should be taken into account while studying any specific algorithm. In one sense, the following are three major difficulties which arise when one uses a decision tree in a real-life situation.

Guillotine Cut

Most decision tree algorithms examine only a single attribute at a time. As mentioned in the earlier paragraph, normally the splitting is done for a single attribute at any stage and if the attribute is numeric, then the splitting test is an inequality. Geometrically, each splitting can be viewed as a plane parallel to one of the axes. Thus, splitting one single attribute leads to rectangular classification boxes that may not correspond too well with the actual distribution of records in the decision space. We call this the *guillotine cut* phenomenon. The test is of the form $(X > z)$ or $(X < z)$, which is called a guillotine cut, since it creates a guillotine cut subdivision of the Cartesian space of the ranges of attributes.

However, the guillotine cut approach has a serious problem if a pair of attributes are correlated. For example, let us consider two numeric attributes, height (in meters) and weight (in Kilograms). Obviously, these attributes have a strong correlation. Thus, whenever there exists a correlation between variables, a decision tree with the splitting criteria on a single attribute is not accurate. Therefore, some researchers propose an *oblique decision tree* that uses a splitting criteria involving more than one attribute.

Overfit

Decision trees are built from the available data. However, the training data set may not be a proper representative of the real-life situation and may also contain noise. In an attempt to build a tree from a noisy training data set, we may grow a decision tree just deeply enough to perfectly classify the training data set.

Definition 6.3 Overfit

A decision tree T is said to overfit the training data if there exists some other tree T' which is a simplification of T, such that T has smaller error than T' over the training set but T' has a smaller error than T over the entire distribution of instances.

Overfitting can lead to difficulties when there is noise in the training data, or when the number of training examples is too small. Specifically, if there is no conflicting instances in the training data set, the error of the fully built tree is zero, while the true error is likely to be bigger. There are many disadvantages of an overfitted decision tree:

a. Overfitted models are incorrect
b. Overfitted decision trees require more space and more computational resources
c. Overfitted models require the collection of unnecessary features
d. They are more difficult to comprehend.

The pruning phase helps in handling the overfitting problem. The decision tree is pruned back by removing the subtree rooted at a node and replacing it by a leaf node, using some criterion. Several pruning algorithms are reported in literature. We shall discuss these algorithms in the latter part of this chapter.

Attribute Selection Error

Most of the decision tree algorithms exhibit a systematic, unwarranted preference for certain types of variables. Some decision tree algorithms are far more likely to construct models that use discrete variables with many values, than discrete variables with relatively few values. This behaviour occurs even though models that use the latter variables have a consistently higher score when tested on a new data set. For example, in a data set, between two attributes, say a state of domicile and gender, the former one which has a larger number of distinct values is preferred over the latter. This behaviour of the algorithm is not desirable.

6.4 Best Split

We have noticed that there are several alternatives to choose from for the splitting attribute and the splitting criterion. But in order to build an optimal decision tree, it is necessary to select those corresponding to the best possible split. The main operations during the tree building are

1. evaluation of splits for each attribute and the selection of the best split; determination of the splitting attribute,
2. determination of the splitting condition on the selected splitting attribute, and
3. partitioning the data using the best split.

The complexity lies in determining the best split for each attribute. The splitting also depends on the domain of the attribute being numerical or categorical. The generic algorithm for the construction of decision trees assumes that the method to decide the splitting attribute at a node and the splitting criteria are known. The desirable feature of splitting is that it should do the best job of splitting at the given stage.

The first task is to decide which of the independent attributes makes the best splitter. The best split is defined as one that does the best job of separating the records into groups, where a single class predominates. To choose the best splitter at a node, we consider each independent attribute in turn.

Assuming that an attribute takes on multiple values, we sort it and then, using some evaluation function as the measure of goodness, evaluate each split. We compare the effectiveness of the split provided by the best splitter from each attribute. The winner is chosen as the splitter for the root node. How does one know which split is better than the other? We shall discuss below two different evaluation functions to determine the splitting attributes and the splitting criteria.

6.5 Splitting Indices

In this section, we shall study two different methods of determining the goodness of a split. One index is based on the information theory, that is, information gain based on entropy. The other one is derived from economics as measure of diversity. This is called the *gini index*.

Entropy

Entropy provides an information-theoretic approach to measure the goodness of a split. Assume that there are n equally probable possible messages. The probability p of each is $1/n$ and hence, the information conveyed by a message is $-\log_2(p) = \log_2(n)$. If there are 16 messages, then since $\log(16) = 4$, we need 4 bits to identify each message. Henceforth, let us assume that all logarithms are in base 2.

Definition 6.4 Entropy

If we are given a probability distribution $P = (p_1, p_2, \ldots, p_n)$, then the information conveyed by this distribution, also called the entropy of P, is

$$\textbf{Entropy}(P) = -[p_1 \log(p_1) + p_2 \log(p_2) + \ldots + p_n \log(p_n)].$$

For example, if P is (0.5, 0.5), then Entropy(P) is 1; if P is (0.67, 0.33), then Entropy(P) is 0.92; if P is (1, 0), then Entropy(P) is 0. Note that the more uniform the probability distribution, the greater is its information.

In the context of decision trees, if the outcome of a node is to classify the records into two classes, C_1 and C_2, the outcome can be viewed as a message that is being generated and the entropy gives the measure of information for a the message to be C_1 or C_2. If a set of records T is partitioned into a set of disjoint exhaustive classes C_1, $C_2, \ldots, C_n$ on the basis of the value of the class attribute, then the information needed to identify the class of an element of T is

$$\text{Info}(T) = \text{Entropy}(P),$$

where P is the probability distribution of the partition C_1, C_2, ..., C_n. P is computed based on their relative frequencies, i.e.,

$$P=\left(\frac{|C_1|}{|T|},\frac{|C_2|}{|T|},\ldots,\frac{|C_n|}{|T|}\right).$$

Suppose that a data set has three distinct classes: C_1, C_2 and C_3. In other words, the class attribute has the domain consisting of $\{C_1, C_2, C_3\}$ and each category has 40, 30, 30 data, respectively. We summarize this in Table 6.4.

Table 6.4

T	C1	C2	C3
100	40	30	30

The value of the entropy of the whole data set is

$$Info(T)=-\frac{40}{100}\log\frac{40}{100}-\frac{30}{100}\log\frac{30}{100}-\frac{30}{100}\log\frac{30}{100}=1.09$$

DEFINITION 6.5 INFORMATION FOR A PARTITION ON X

If T is partitioned based on the value of the non-class attribute X, into sets T_1, T_2, ..., T_n, then the information needed to identify the class of an element of T becomes the weighted average of the information to identify the class of the element of T_i, i.e., the weighted average of $Info(T_i)$

$$Info(X,T)=\sum_{i=1}^{n}\frac{|T_i|}{|T|}Info(T_i).$$

Let us consider splitting the data set into two subsets, S_1 and S_2, with n_1 and n_2 data, respectively, where $n_1 + n_2 = n$. If we assume $n_1 = 60$ and $n_2 = 40$, the splitting is as follows (Table 6.5):

Table 6.5a

S2	C1	C2	C3
40	0	20	20

Table 6.5b

S1	C1	C2	C3
60	40	10	10

The entropy index value of the data set after the segmentation is

$$\frac{40}{100}\left(-\frac{20}{40}\log\frac{20}{40}-\frac{20}{40}\log\frac{20}{40}\right)+\frac{60}{100}\left(-\frac{40}{60}\log\frac{40}{60}-\frac{10}{60}\log\frac{10}{60}-\frac{10}{60}\log\frac{10}{60}\right)=0.80.$$

DEFINITION 6.6 GAIN

We define the information gain due to a split on attribute X as

$$Gain(X, T) = Info(T) - Info(X, T).$$

The information gain represents the difference between the information needed to identify an element of T and the information needed to identify an element of T after the value of attribute X is obtained. That is, the information gain due to attribute X.

In the above example, splitting decreases the value of the entropy by 0.29. In other words, the gain is 0.29.

Let us consider another splitting as given in Table 6.6.

In this case, the value of the associated entropy is 1.075 and the gain is 0.015. Thus, we can use this notion of gain to rank attributes and to build decision trees where, at each node, the attribute with greatest gain becomes the splitting attribute.

Table 6.6a

S1	C1	C2	C3
60	20	20	20

Table 6.6b

S2	C1	C2	C3
40	20	10	10

EXAMPLE 6.3

In our earlier example on playing, we had

$$Info(T) = Entropy\left(\frac{9}{14}, \frac{5}{14}\right) = -\frac{9}{14}\log\frac{9}{14} - \frac{5}{14}\log\frac{5}{14} = 0.94.$$

Consider the attribute *outlook*. It has three distinct values—sunny, overcast and rain, with 5, 4 and 5 records, respectively. Among the 5 records which have *outlook* = *sunny*, 2 records are in the *play* class and 3 are in the *no play* class. Thus, these five records have a distribution (3/5, 2/5). Similarly, the distributions for *overcast* and *rain* are (1, 0) and (3/5, 2/5), respectively. Thus,

$$\begin{aligned} Info(oulook, T) &= \sum_{i=1}^{3} \frac{|T_i|}{|T|} Info(T_i) \\ &= \frac{5}{14} Info\left(\frac{3}{5}, \frac{2}{5}\right) + \frac{4}{14} Info(1, 0) + \frac{5}{14} Info\left(\frac{3}{5}, \frac{2}{5}\right) \\ &= \frac{5}{14}\left(-\frac{3}{5}\log\frac{3}{5} - \frac{2}{5}\log\frac{2}{5}\right) + \frac{5}{14}\left(-\frac{3}{5}\log\frac{3}{5} - \frac{2}{5}\log\frac{2}{5}\right) \\ &= 0.694. \end{aligned}$$

Thus,

$$Gain(outlook,T) = Info(T) - Info(outlook,T) = 0.94 - 0.694 = 0.246.$$

Let us consider the attribute *humidity*. If we take all the distinct numeric values of this attribute, then there are 9 values – 65, 70, 75, 78, 80, 85, 90, 95 and 96 – with frequencies 1, 3, 1, 1, 3, 1, 2, 1 and 1, respectively.

Thus,

$$Info(humidity,T) = \frac{3}{14}\left(-\frac{2}{3}\log\frac{2}{3} - \frac{1}{3}\log\frac{1}{3}\right) + \frac{3}{14}\left(-\frac{2}{3}\log\frac{2}{3} - \frac{1}{3}\log\frac{1}{3}\right) + \frac{2}{14}\left(-\frac{1}{2}\log\frac{1}{2} - \frac{1}{2}\log\frac{1}{2}\right) = 0.5364.$$

$$Gain(humidity,T) = Info(T) - Info(humidity,T) = 0.94 - 0.5364 = 0.4036.$$

So, the gain due to selecting *humidity* as the splitting attribute is high. But the number of branches are many—one branch for each distinct value; this is not practical (Note: this may lead to overfitting too). Let us assume that we categorize these values into two sets, *high* and *low*, such that the value exceeding 75 is said to be of high humidity or else, it is of low humidity. There are 9 records with high humidity, of which 4 are in the *no play* class and 5 are in the *play* class. Similarly, out of the 5 records of low humidity, 1 is in the *no play* class and the remaining 4 are in the *play* class. In such a situation, the gain is calculated as follows:

$$Info(humidity,T) = \frac{9}{14}\left(-\frac{4}{9}\log\frac{4}{9} - \frac{5}{9}\log\frac{5}{9}\right) + \frac{5}{14}\left(-\frac{4}{5}\log\frac{4}{5} - \frac{1}{5}\log\frac{1}{5}\right) = 0.8435.$$

Hence, the gain is 0.0965.

For the attribute *windy*, we have the following expression:

$$Info(windy,T) = \frac{6}{14}\left(-\frac{3}{6}\log\frac{3}{6} - \frac{3}{6}\log\frac{3}{6}\right) + \frac{8}{14}\left(-\frac{6}{8}\log\frac{6}{8} - \frac{2}{8}\log\frac{2}{8}\right) = 0.892.$$

Thus, the gain is 0.082.

For the attribute *temperature*, we have

$$Info(temp,T) = \frac{8}{14}\left(-\frac{5}{8}\log\frac{5}{8} - \frac{3}{8}\log\frac{3}{8}\right) + \frac{6}{14}\left(-\frac{4}{6}\log\frac{4}{6} - \frac{2}{6}\log\frac{2}{6}\right) = 0.939.$$

Thus, the gain is 0.001.

Splitting on *outlook* provides the maximum gain. The root node is expanded with three child nodes, one for each value of *outlook*. The data set is also partitioned into

three subsets—one subset is associated with each child node. The algorithm proceeds recursively till the termination criterion is met.

Gain Ratio

The notion of gain, introduced earlier, tends to favour attributes that have a large number of values. For example, if we have an attribute X that has a distinct value for each record, then *Info*(*X*,*T*) is 0, thus *Gain*(*X*,*T*) is maximal. We noticed this for *humidity* above. To compensate for this, Quinlan suggests using the following gain-ratio instead of gain.

Definition 6.6 Gain Ratio

$$Gain\_ratio(X,T) = \frac{Gain(X,T)}{Info(X,T)}.$$

For example,

$$Gain\_ratio(outlook,T) = \frac{Gain(outlook,T)}{Info(outlook,T)} = \frac{0.246}{0.694} = 0.3544.$$

Gini Index

One of the goodness measures can be related to measure of diversity. There are several ways of calculating the index of diversity for a set of records. With all of them, a high index of diversity indicates that the set contains an even distribution of classes; while a low index means that members of a single class predominates. The best splitter is the one that decreases the diversity of the record sets by the greatest amount, in other words, we want to maximize

diversity(before split) – (diversity(left child) + diversity(right child)).

The gini index is a diversity measure from economics. It can also be used to evaluate the goodness of a split.

Definition 6.7 Gini Index

If a data set *T* contains *n* classes, then *gini(T)* is defined as

$$gini(T) = 1 - \sum p_i^2.$$

where p_j is the relative frequency of class *j* in *T*. If the split divides *T* into T_1 and T_2, then the index of the divided data is given as

$$gini_{spli}(T) = \frac{n_1}{n} gini(T_1) + \frac{n_2}{n} gini(T_2).$$

In the above example,

$$gini(T) = 1 - \left[\frac{9}{14}\right]^2 - \left[\frac{5}{14}\right]^2 = 0.46.$$

Thus, the gini index due to splitting on *outlook* is

$$gini_{outlook}(T) = \frac{5}{14}\left[1 - \left[\frac{3}{5}\right]^2 - \left[\frac{2}{5}\right]^2\right] + \frac{4}{14}[1-1] + \frac{5}{14}\left[1 - \left[\frac{3}{5}\right]^2 - \left[\frac{2}{5}\right]^2\right] = 0.343.$$

The best splitter is determined as the attribute which has the smallest gini value.

6.6 Splitting Criteria

While determining the splitting attribute, we should keep in mind the splitting criterion too. For instance, some algorithms generate only binary decision trees. In other words, these algorithms carry out only binary splits. In case the domain of the splitting attribute is binary this does not pose any problem, as the attribute itself takes two distinct values only. But, on the other hand, if the attribute that is selected for splitting takes more than two values, or takes continuous values, or is a categorical attribute, a binary split necessarily implies that we must determine the splitting criterion. Please note that the choice of the splitting criterion cannot be a different process from that of determining splitting attributes. It is a part of the latter, as the gain index is calculated after we know the resulting partition of the data set. For a non-binary split, most algorithms partition the data set for all distinct values. However, some algorithms (mostly the AID families of algorithms) have a variable split size. We shall discuss below some strategies for binary splits for numerical and categorical attributes.

Binary Splits for Numerical Attributes

The numerical attributes are split by the binary split of the form $A \leq v$, where v is a real number. All the numerical attributes are sorted on the values of the attribute being considered for splitting. Let $A_1, A_2, \ldots, A_n$ be the sorted values of a numerical attribute A. Since any value between A_i and A_{i+1} divides the data into two subsets, we need to examine only $n-1$ possible splits. The midpoint of A_i and A_{i+1} is taken as the split point. Thus, while computing the splitting index (gain or gini index), we take into account these candidate split points and determine the split points corresponding to the best split.

CLASS HISTOGRAM

We have noted in the foregoing discussion that in order to compute the splitting indices, whether it is entropy-based gain or gini index, we need the frequency distribution of class values for the attribute under consideration. This distribution is represented as a *class histogram.*

DEFINITION 6.8 CLASS HISTOGRAM

Let the training data set be T with class labels $\{C_1, C_2, \ldots, C_k\}$. If T is partitioned based on the value of the non-class attribute X into sets $T_1, T_2, \ldots, T_n$, then the class histogram for the partition is a table of k columns and n rows. The (i, j)-th entry indicates the number of records in the data set in the partition T_i and class C_j.

For a numerical attribute A, let us assume that we want to compute the splitting index for the possible split $A \leq v$. Then, the class histogram is a table of two rows and k columns, where k is the number of classes. The first row represents the frequency distribution of the set of records satisfying $A \leq v$. The second row represents the frequency distribution for each class, for those records which do not satisfy the condition.

EXAMPLE 6.3

Consider the following data (Table 6.7) and the class histogram (Table 6.8) for the attribute 'Salary', when the possible splitting criterion is Salary ≤ 70. We can see that out of 4 records satisfying this condition there are 2 records in class '*B*' and two in class '*G*'. This is indicated in first row indicated as 'left', denoting the left child node

Table 6.7 Training Data Set

	Age	Salary	Class
1	30	65	G
2	23	15	B
3	40	75	G
4	55	40	B
5	55	100	G
6	45	60	G

Table 6.8 Class Histogram for S*alary* with Splitting Criterion 'Salary ≤ 70'.

SALARY ≤ 70	B	G
left	2	2
right	0	2

of the current node. Similarly, the right child node corresponds to the second row of the table.

Please note that the class histogram contains all the information for the computation of the splitting index. For the following generic histogram, the gini index is given below.

	C1	C2
L	a1	a2
R	b1	b2

$$gini = \frac{(a1+a2)}{n}\left[1-\left(\frac{a1}{a1+a2}\right)^2-\left(\frac{a2}{a1+a2}\right)^2\right]+\frac{(b1+b2)}{n}\left[1-\left(\frac{b1}{b1+b2}\right)^2-\left(\frac{b2}{b1+b2}\right)^2\right]$$

This concept can be generalized for the cases where the split is non-binary. In such cases, the number of rows is equal to the number of partitions. Hence, one can also define a class histogram for a categorical attribute[1].

Binary Splits for Categorical Attributes

The splitting point for categorical attributes is different. Since we cannot have any ordering of the values of a categorical attribute, there cannot be a value n such that it splits the attribute into two. If $S(A)$ is the set of possible values of the categorical attribute A, then the split test is of the form $A \in S'$ where $S' \subset S$. For an attribute with n values, there are 2^n possible splits. If n is small, the split index value is found for all the possible combinations and the best split is taken. If n is large, then the split is made by some heuristics and the best split among them is found. The construction of an attribute list is similar to that of numerical attributes. But instead of having a class histogram, a count matrix is maintained for the categorical attribute.

Definition 6.9 Count Matrix

The count matrix has n rows (for n distinct values of the attribute) and k columns (for k classes). Each entry, say (i, j)-entry, represents the number of records in the data set having i^{th} value of the attribute and in the j^{th} class.

Note that the count matrix is independent of any partition, whereas the class histogram is specific to a partition. Different splitting criteria result in different class histograms.

[1] When such a table is constructed for all distinct values of an attribute, it is called the Count Matrix. This is useful in the context of categorical attributes.

EXAMPLE 6.4

The following table (Table 6.10) illustrates the count matrix for the data given in Table 6.9.

Table 6.9 Attribute List for a Categorical Attribute

	Class	record_id
family	high	1
sports	high	2
sports	high	3
family	low	4
truck	low	5
family	high	6

Table 6.10 Count Matrix

	H	L
family	2	1
sports	2	0
truck	0	1

The count matrix contains all the information necessary for determining the splitting index.

For example, if we select $S' = \{\text{family, truck}\}$ as the possible splitting subset, then the gini index is given by

$$gini_{S'}(T) = \frac{4}{6}\left[1-\left(\frac{2}{4}\right)^2-\left(\frac{2}{4}\right)^2\right]+\frac{2}{6}\left[1-\left(\frac{2}{2}\right)^2-\left(\frac{0}{2}\right)^2\right]=\frac{1}{3}.$$

Such a calculation is to be carried out for each subset of {family, truck, car}, to determine the splitting criteria.

6.7 DECISION TREE CONSTRUCTION ALGORITHMS

A number of algorithms for inducing decision trees have been proposed over the years. These algorithms more or less follow the principles discussed above. However, they differ among themselves in the methods employed for selecting splitting attributes and splitting conditions. In the following few sections, we shall study some of the major methods of decision tree constructions. We categorize these algorithms into two types. The first type of algorithms is the classical algorithms which handle only memory-resident data. Efficiency and scalability are the fundamental issues concerning the

classification of data. The second category of algorithms described below can handle the efficiency and scalability issues. But they are, in a sense, derivatives of the first category. These algorithms remove the memory restrictions and are fast and scalable. They also present a unifying framework for decision tree classifiers, that separate the scalability aspects of the algorithm for constructing a decision tree from the central features that determine the quality of the tree. In this category, we shall study two exact algorithms namely, SPRINT and RainForest. We shall also study some approximate algorithms like BOAT and CLOUDS.

6.8 CART

CART (**C**lassification **A**nd **R**egression **T**ree) is one of the popular methods of building decision trees in the machine learning community. CART builds a binary decision tree by splitting the records at each node, according to a function of a single attribute. CART uses the gini index for determining the best split. CART follows the above principle of constructing the decision tree. We outline the method for the sake of completeness

The initial split produces two nodes, each of which we now attempt to split in the same manner as the root node. Once again, we examine all the input fields to find the candidate splitters. If no split can be found that significantly decreases the diversity of a given node, we label it as a leaf node. Eventually, only leaf nodes remain and we have grown the full decision tree. The full tree may generally not be the tree that does the best job of classifying a new set of records, because of overfitting.

At the end of the tree-growing process, every record of the training set has been assigned to some leaf of the full decision tree. Each leaf can now be assigned a class and an error rate. The error rate of a leaf node is the percentage of incorrect classification at that node. The error rate of an entire decision tree is a weighted sum of the error rates of all the leaves. Each leaf's contribution to the total is the error rate at that leaf multiplied by the probability that a record will end up in there.

6.9 ID3

Quinlan introduced the ID3, **I**terative **D**ichotomizer 3, for constructing the decision trees from data. In ID3, each node corresponds to a splitting attribute and each arc is a possible value of that attribute. At each node the splitting attribute is selected to be the most informative among the attributes not yet considered in the path from the root. Entropy is used to measure how informative is a node. This algorithm uses the criterion of information gain to determine the goodness of a split. The attribute with the greatest information gain is taken as the splitting attribute, and the data set is split for all distinct values of the attribute.

6.10 C4.5

C4.5 is an extension of ID3 that accounts for unavailable values, continuous attribute value ranges, pruning of decision trees and rule derivation. In building a decision tree, we can deal with training sets that have records with unknown attribute values by evaluating the gain, or the gain ratio, for an attribute by considering only those records where those attribute values are available. We can classify records that have unknown attribute values by estimating the probability of the various possible results. Unlike CART, which generates a binary decision tree, C4.5 produces trees with variable branches per node. When a discrete variable is chosen as the splitting attribute in C4.5, there will be one branch for each value of the attribute.

6.11 CHAID

CHAID, proposed by Kass in 1980, is a derivative of AID (Automatic Interaction Detection), proposed by Hartigan in 1975. CHAID attempts to stop growing the tree before overfitting occurs, whereas the above algorithms generate a fully grown tree and then carry out pruning as post-processing step. In that sense, CHAID avoids the pruning phase.

In the standard manner, the decision tree is constructed by partitioning the data set into two or more subsets, based on the values of one of the non-class attributes. After the data set is partitioned according to the chosen attributes, each subset is considered for further partitioning using the same algorithm. Each subset is partitioned without regard to any other subset. This process is repeated for each subset until some stopping criterion is met. In CHAID, the number of subsets in a partition can range from two up to the number of distinct values of the splitting attribute. In this regard, CHAID differs from CART, which always forms binary splits, and from ID3 or C4.5, which form a branch for every distinct value.

The splitting attribute is chosen as the one that is most significantly associated with the dependent attributes according to a chi-squared test of independence in a contingency table (a cross-tabulation of the non-class and class attribute). The main stopping criterion used by such methods is the p-value from this chi-squared test. A small p-value indicates that the observed association between the splitting attribute and the dependent variable is unlikely to have occurred solely as the result of sampling variability.

If a splitting attribute has more than two possible values, then there may be a very large number of ways to partition the data set based on these values. A combinatorial search algorithm can be used to find a partition that has a small p-value for the chi-squared test. The p-values for each chi-squared test are adjusted for the multiplicity of partitions. A Bonferroni adjustment is used for the p-values computed from the

contingency tables, relating the predictors to the dependent variable. The adjustment is conditional on the number of branches (compound categories) in the partition, and thus does not take into account the fact that different numbers of branches are considered.

6.12 Summary

The above algorithms (mainly two) form the basis of all the decision tree construction for data mining. We can use either entropy-based information gain or the gini index as the splitting index. We can use a binary split or a variable number of splits. The recent algorithms proposed specifically for data mining are based on the basic principles discussed above. However, the recent proposals are more suitable for very large databases. There are two different ways of handling large databases for the construction of decision trees.

One way is to construct the tree on a smaller subset of data. Quinlan proposes a 'windowing' technique. Others propose sampling the data set. We shall discuss this aspect later. The disadvantage of this process is that these approaches try to compromise on accuracy. The accuracy of the tree constructed from the whole training set is better than that of the tree constructed from a subset of the training set.

The second approach is to improve upon the construction method without any relaxation on the accuracy. It can seen that while determining the best split at every node, a major computational overhead is sorting. At every node, the attribute values of each attribute are arranged in an order to determine the class histogram and the splitting index. For a large database, this turns out to be a very expensive method. The major question that comes to mind is: "Can we avoid this repeated sorting?'. In the following section, we shall discuss a novel algorithm which avoids such repeated sorting.

6.13 Decision Tree Construction with Presorting

The researchers on the QUEST project at IBM (Rakesh Agrawal and team) proposed two algorithms SLIQ (Supervised Learning In Quest) and SPRINT (Scalable Parallelizable Induction of Decision Tree) in sequel. The latter one, SPRINT, is aimed at parallelizing the older one, SLIQ. However, SLIQ has very interesting and novel features. SLIQ imposes no restrictions on the amount of training data or the number of attributes in the examples. We shall now discuss the novel features proposed by SLIQ and SPRINT. The algorithm presented here is very close to SPRINT but is not exactly the same as the original proposal of SPRINT. The additional features of the original SPRINT that are different from the present algorithm are discussed in the next section.

SLIQ (and SPRINT) is a scalable algorithm, which uses a pre-sorting technique integrated with a breadth-first tree growing strategy for the classification of the disk-resident data. It builds a decision tree in a breadth-first manner and avoids repeated sorting at every step. The choice of the splitting criterion depends on the domain of the attribute being numeric or categorical.

DEFINITION 6.10 ATTRIBUTE LIST

Let the training data set be T with class-labels $\{C_1, C_2, \ldots, C_k\}$, and let X be a non-class attribute of T. An attribute list of X with respect to T, is a table with three columns. The columns corresponds to *attribute value, class label,* and *record_id (rid)* of the record of T from which these values are obtained. The number of rows in the attribute list is same as the number of tuples in T. If X is a numeric attribute, then the attribute list is sorted on the field *attribute value*

Note that the attribute list is sorted by the attribute values only once, when it is first created. The attribute lists can be disk-resident, if necessary.

DEFINITION 6.11 RECORD LIST

A record list is a table which has two columns: the first one contains the record id (*rid*), and the second one is the *node number* of the tree to which the record is currently allocated.

Recall that every node corresponds to a subset of the data set. In other words, the record list can at any time identify the partition to which a record belongs. Thus, during the construction, any record is allocated to a node—initially to the root. In this list, the rids appear in the same order as the records appear in the original data set. This structure is required to be randomly accessed and hence is memory-resident.

The concept is illustrated through the following examples.

EXAMPLE 6.5

Table 6.11 contains six records and three attributes. One extra column is introduced to show the record number. The attributes 'Age' and 'Salary' are non-class attributes. So,

Table 6.11 Another Sample Data Set

rid	Age	Salary	Class
1	30	65	G
2	23	15	B
3	40	75	G
4	55	40	B
5	55	100	G
6	45	60	G

Table 6.12a Attribute List for Age

Age	Class	rid
23	B	2
30	G	1
40	G	3
45	G	6
55	G	5
55	B	4

Table 6.12b Attribute List for Salary

Salary	Class	rid
15	B	2
40	B	4
60	G	6
65	G	1
75	G	3
100	G	5

we have two attribute lists, one for each of these two attributes. Tables 6.12 show the attribute list corresponding to Age and Salary. Note that in an attribute list, the values are arranged in the sorted order of the values of the attribute. Table 6.13 shows the record list at the root.

Table 6.13 Record List(at root)

rid	Leaf
1	N1
2	N1
3	N1
4	N1
5	N1
6	N1

The advantage of the attribute list structure is that it does not require repeated sorting. The initial lists created from the training data set are associated with the root node. As the tree grows, and the data sets are partitioned, the order of records in the attribute list is preserved. Thus, the new partitions at the child nodes do not require any further sorting. The major steps of determining the split are

- splitting index for each value of each attribute,
- determining the best split, and
- partitioning the data set of the current node and allocate to each child node.

Computation of Splitting Index

The gini index is used as the goodness measure. The attribute lists come in very handy to compute the class histogram. In fact, one needs to have just one pass over the attribute list for determining the splitting point of an attribute.

The splitting index of each valuc of a given attribute is computed by one pass over the attribute list. The attribute list is never partitioned at any stage as a result of the construction process, we have only one list per attribute to be scanned. The advantage is that we can make just one scan of an attribute list to evaluate the split index of this attribute throughout the tree (at each of the current leaf nodes of the tree). This is the motivation for expanding the tree in a breadth-first manner. The actual partitioning is recorded only in the record list which is memory resident. This record list is updated at cvery split.

For Numerical Attributes

For numeric attributes, the candidate split point is taken to be the midpoint of two consecutivc distinct values. We read each tuple of the attribute list one by one. Initially, thc class histogram has all the records in the second row, that is the 'right' row. All the entries in 'left' row are 0. After reading each tuple, the histogram is updated for the new value of the attribute. Since the attribute list is sorted, the histogram for the next tuple can be updated using the previous values. Thus, the gini values for all the attributc lists can be computed in one pass and the best split point can bc found. As the attribute list is not physically split, the process of computation of the gini index for intermediate nodes is only slightly different. This is because the root node corresponds to the whole data set, whereas the intermediate nodes represent only a subset of the data set. The tree is expanded in a breadth-first manner and hence, all the current-leaf nodes are processed at any stage. For each attribute, the corresponding attribute list is scanned once for the given level to evaluate the splitting indices for all the nodes at that level. We illustrate this process for our example given in Table 6.11.

At the Root Node

The histogram for 'Salary' is intialized as follows. This is before reading any tuple of the attribute list associated with 'Salary'. The gini index can also be calculated from this histogram.

Table 6.14

Salary < 15	B	G
L	0	0
R	2	4

The histogram is updated after the first record of the attribute list is read. It is observed that the first record is in class 'B'. The splitting value is taken as 28, the midpoint of 15 and 40.

Table 6.15

Salary < 28	B	G
L	1	0
R	1	4

The second tuple is read and it is also in class 'B' and thus, we get the updated histogram as follows. We give below the sequence of updates of the histogram.

Table 6.16

a

Salary < 50	B	G
L	2	0
R	0	4

b

Salary < 63	B	G
L	2	1
R	0	3

c

Salary < 70	B	G
L	2	2
R	0	2

d

Salary < 88	B	G
L	2	3
R	0	1

Note that these tables carry adequate information to evaluate the splitting criteria for the salary attribute. This process is to be carried out for each of the attributes.

At Any Intermediate Node

Let us assume that we are at the second level of the tree. The root node has been expanded with an initial split of *age* ≤ 35. Thus, the updated record list is as follows (Table 6.17). The records 1 and 2 are allocated to node N2 (left child node of the root node), and the remaining records are allocated to N3 (right child node).

In order to determine the splitting index for each value of the attribute 'Salary' for each of the nodes N2 and N3, we again make one more pass over the attribute list of 'Salary'. At this stage, we maintain two class histograms—one for each of the leaf nodes. Before reading the first tuple, the class histograms are initialized as follows (Table 6.18).

Table 6.17

rid	Leaf
1	N2
2	N2
3	N3
4	N3
5	N3
6	N3

Table 6.18 Class Histograms for Child Nodes of N1 at the beginning

Salary < 15	B	G
L	0	0
R	1	1

Salary < 15	B	G
L	0	0
R	1	3

After reading the first tuple of the attribute list,

Table 6.19 Class histogram for Child Nodes of N1

Salary < 28	B	G
L	1	0
R	0	1

Salary < 28	B	G
L	0	0
R	1	3

After reading the second tuple of the attribute list,

Table 6.20

Salary < 50	B	G
L	1	0
R	0	1

Salary < 50	B	G
L	1	0
R	0	3

After reading the third record, we have

Table 6.21

Salary < 70	B	G
L	1	1
R	0	0

Salary < 70	B	G
L	1	1
R	0	2

When we read the fourth record, the new class histograms are

Table 6.22

Salary < 88	B	G
L	1	1
R	0	0

Salary < 88	B	G
L	1	2
R	0	1

After the fifth record, we have

Table 6.23

Salary < 88	B	G
L	1	1
R	0	0

Salary < 88	B	G
L	1	2
R	0	1

After reading the last tuple,

Table 6.24

Salary > 100	B	G
L	1	1
R	0	0

Salary > 100	B	G
L	1	3
R	0	0

For Categorical Attributes

The process of finding splitting criteria is different for categorical attributes. In the case of numerical attributes, after every tuple of the attribute list is read, the histogram is updated. In the case of categorical attributes, we make a complete scan through the attribute list collecting the counts in the histogram (it is called the *count matrix* for categorical attributes). Once we are finished with the scan, the count matrix is used to find the split points. If the number of distinct values is reasonably small, we consider all the subsets of attribute values as possible split points and compute the corresponding gini index. If there are n values, it amounts to checking 2^n possibilities. If n exceeds a threshold, a greedy algorithm can be used. In the greedy method, we start with an empty subset S' and add the element of S (domain of categorical attribute A) that gives the maximum decrease in the gini index. The process is repeated by adding one element at every step, until there is no reduction in the gini index.

Example 6.6

An example is shown below (Table 6.25) which indicates the histogram construction for the categorical attribute.

Table 6.25 Histogram for the Categorical Attribute

	Class	rid
family	high	1
sports	high	2
sports	high	3
family	low	4
truck	low	5
family	high	6

Table 6.26 Count Matrix

	H	L
family	2	1
sports	2	0
truck	0	1

One way to calculate the splitting criteria is to check the gini index for all 8 subsets of {family, sports, truck} and decide the splitting condition based on the minimum gini index. All the distinct values are taken with their class distributions. In the given example, all the possible combinations are [{family}, {sports,truck}], [{family, sports}, {truck}], [{family, truck}, {sports}]. In case the set is very large, checking all subsets is not practical. In such a situation, we employ some sort of heuristics. One popular method is to use a greedy method for this purpose. We demonstrate the working of the greedy method for the above example.

Greedy Method to Determine the Splitting Point of Categorical Attributes

We begin with the empty subset, $S' = \varnothing$. Hence, the gini index is 2/3. We now check each element to see if adding any element to S' results in a decrease in the gini index

and, if there are such elements, then to choose the element that decreases the index value to the maximum. Let us assume that $gini_{S'}(T)$ represents the gini index for the possible split with S'.

Thus, $gini_{\{family\}}(T) = 4/9$, $gini_{\{sport\}} = 1/3$ and $gini_{\{truck\}} = 4/15$. Thus, when S' is either {sport} or {truck}, the gini decreases. But the maximum decrease is due to {truck}. Thus, S' is updated as {truck}. In the next step, we compute $gini_{\{truck, family\}} = 1/3$ and $gini_{\{truck, sport\}} = 8/9$. None of these results in a decrease in the gini index. Hence, the construction process stops here and the splitting criteria is based on $S' = \{truck\}$.

Updating the Record List

Once the best split is found, the next step is to partition the data, create the child nodes for each leaf node and update the record list. Recall that the record list depicts the allocations of records to nodes. Since at any split, nodes are expanded and data sets are partitioned, the records are allocated to newly-generated leaf nodes. This change should be incorporated in the record list. The algorithm for updating the class list is given below.

For each attribute A used in the split, traverse the attribute list of A. At each value n in the attribute list, using *rid* as the reference field, find the corresponding entry (node number) in the record list, (say e). Find the node number of a new child node, say f, to which n belongs, by applying the splitting test corresponding to node e. Update the record list replacing e by f.

Example 6.5 (contd.)

Consider the example shown in Table 6.11. Let us suppose that the best split is found for $salary \leq 50$. The salary attribute is traversed and the class list entries for the leaf, corresponding to the class list indexes for $salary \leq 50$, are updated. Here, the class list indexes for $salary \leq 50$ are 2 and 4 and so the leaf node of the record list corresponding to the entries 2 and 4 are updated to the new leaf node N2 and the other entries of the leaf are updated to N3. The above two steps of splitting the nodes and updating the leaf nodes are repeated until each leaf node contains records belonging to the same class and thus, no further splits are needed.

Table 6.27 Updating the Class List

rid	Leaf
1	N3
2	N2
3	N3
4	N2
5	N3
6	N2

SPRINT

SPRINT is the updated version of SLIQ and is meant for parallel implementation. In SPRINT, the attribute list is partitioned at each split, whereas in SLIQ a single attribute list is maintained for an attribute. Once the split point is found, the split is done by creating the child nodes and partitioning the attribute records between the child nodes. The attribute lists for all the attributes are split at each node among the child nodes. Splitting the winning attribute, i.e., the attribute list that has the split point, is straightforward. All the attribute records above the split point are moved to the left child node, while the other records are moved to the right child node. For the other attribute lists, we use the 'rid' field of the attribute list. While splitting the winning attribute, the record list is referred to note the child node to which the record is allocated. SPRINT, instead of maintaining a record list, keeps this information in a hash table.

Please remember that the basic principle of SLIQ and SPRINT is independent of the evaluation function. One can use the entropy-based method also. The basic idea of the above algorithm is to avoid repeated sorting, by maintaining an attribute list for each attribute. This principle works well, irrespective of which splitting index is used. We can replace the computation of the gini index by an entropy-based information gain. As long as the splitting index can be computed from the class histogram, the main steps of the algorithm remain unchanged.

6.14 RainForest

RainForest, like other algorithms, is a top-down decision tree induction scheme. It is similar to SPRINT, but instead of attribute lists it uses a different structure called the AVC-set. The attribute list has one tuple corresponding to each record of the training data set. But the AVC-set has one entry for each distinct value of an attribute. Thus, the size of the AVC-set for any attribute is much smaller.

Definition 6.12 AVC-set

The AVC *(Attribute Value, Class)*-set of a non-class attribute X at a node n, is defined as set of distinct values of X in the records allocated to the node n, and the class label, whereby the individual class labels are aggregated.

The *AVC-group* of a node n is defined to be the set of all AVC-sets for all attributes at node n.

The size of the AVC-set of an attribute depends only on the number of distinct attribute values of X and the number of class labels at node n. It is observed that for most real-life data sets, the whole AVC-group of the root node fits into the main memory. If not, at least the AVC-set of each attribute fits into the main memory. The AVC-set contains the aggregated information that is sufficient for the decision tree construction. The RainForest algorithm proceeds in the same way as SPRINT.

There are different versions of the RainForest algorithm. Depending on the amount of main memory available, three cases can be distinguished.

The AVC-group of the Root Node Fits in the Main Memory

This version of RainForest is called RF-Write. This algorithm assumes that the AVC-group of the root node fits into the main memory. The database is scanned and the AVC-group of the root node is constructed. The standard classification algorithm is applied and the split point is found. The database is scanned and each tuple is written to one of the child nodes, based on the split point. The algorithm then recurses in turn on each of the partitions, till all the class labels in a partition belong to the same class. For each level of the tree, the entire database is read twice and written once.

The AVC-set Fits into Main Memory

Each individual AVC-set of the root node fits into the main memory, but the AVC-group of the root node does not fit into the main memory. It is called the *RF-Read* version. This algorithm reads the original database instead of writing the partitions for the child nodes. When the AVC-group of the child nodes does not fit into the memory, the database is read many times to construct the AVC-groups for an unexamined subset of the new nodes in the tree.

The individual AVC-sets are Too Large

This method is called *RF-Hybrid.* The method proceeds like RF-Read, whenever a level of the tree L has all the AVC-groups fitting into the main memory. At a particular level, when the main memory is not sufficient, it switches to RF-Write.

6.15 Approximate Methods

Another way of handling a large database for the construction of a decision tree is to adopt an approximate method, contrary to the above methods which are exact methods. There are two different approaches for devising approximate constructions of decision trees. These are: (a) Discretization, and (b) Sampling. Discretization reduces the size of the database by grouping the attribute values to a smaller set. Sampling selects a subset of the training data set and thus operates on a smaller data set. Since the original training data set is tampered with in any of these processes, accuracy is the most important measure of effieiency for these algorithms.

Windowing

The windowing technique is useful to construct decision trees on very large data sets. A subset of the training set, called *window*, is chosen at random and a decision tree is constructed using only this window. All other records in the training data set are subjected to classification using this tree. If the tree gives the correct classification for the entire training data set, the process terminates. If not, a selection of the incorrectly classified object is added to the window, and the process continues.

Attribute-oriented Induction

Attribute-oriented induction is another way of discretization. The training data set is compressed with a *concept tree ascension* technique, which is similar to moving up in the dimension hierarchy (discussed in Chapter 2). By this technique, the individual attribute values are replaced by some generalized value provided in the higher level of the concept tree. For example, rather than having individual readings of humidity measures, we can just have high and low humidity. The concepts 'high' and 'low' are higher in the level in the concept tree than 75%, 35% , etc. In this process, the data set contains generalized tuples, and each generalized tuple has a count associated with it. The count corresponds to the number of tuples in the original data set that a generalized tuple represents. In addition to substantially reducing the size of the training set, attribute-oriented induction allows the rules to be comprehensible. The concept hierarchy is generally obtained from the individual application domain. For numerical attributes, such hierarchy can also be generated automatically from the data. It is worthwhile to work out the method of maintaining a class histogram in such cases.

Attribute Removal

Attribute-oriented induction can also be useful for removing some attributes from the set of candidate splitting attributes. If an attribute has a large number of distinct values and there is no higher concept for it, then the attribute can be ignored for decision tree construction.

Another technique of attribute removal can be based on *relevance analysis*. The information theoretic measure for relevance is the uncertainty coefficient. One can retain only the top h most relevant attribute or retain only those attributes whose relevance exceeds a specified threshold. The uncertainty coefficient is defined as follows.

Definition 6.13 Uncertainty Coefficient

The *uncertainty coefficient* for an attribute X is given by

$$UC(X) = \frac{gain(X,T)}{Info(T)}.$$

6.16 CLOUDS

CLOUDS (Classification of Large or Out-of-core Data Sets) is a kind of approximate version of the SPRINT method. It also uses a breadth-first strategy to build the decision tree. CLOUDS uses the gini index for evaluating the split index of the attributes. There are different criteria for splitting the categorical and numerical attributes. The method of finding the gini index for the categorical attributes is the same as that used in the SPRINT algorithm. Categorical attributes do not require any sorting, but it refers to the count matrix. We discuss the steps in which CLOUDS differs from the SPRINT method.

Coarse Splitting Points for Numeric Attribute

A random sample S of size s is drawn from the data set. This requires one read operation of the whole data set, only at the root level. The sample S is used to divide the values of any attribute into q intervals by regular sampling. This corresponds to q regularly-spaced quantiles of the set S. The range of i^{th} interval is $[z_{i-1}, z_i)$; z_0 and z_q are the minimum and the maximum values of the attribute in S. We construct the class histogram for these intervals (note that it differs from the earlier method, where we store the class frequency with respect to each value of the attribute). In order to generate the class frequency for any attribute with respect to these intervals, we need to have an additional database pass. For each value of an attribute in class i, determine the corresponding interval using the binary search and update the class frequency of class i.

At this stage, the gini index is evaluated at each interval boundary and the minimum value of the gini index among all the intervals and across all the numeric attributes is found, which is given by gini_{min}. This is taken as a measure of the splitting criterion for numerical attributes. The split point with the gini_{min} is used as the splitter.

The splitting criterion is applied to each record of S to determine its partition, i.e., left or right. For the node i at level l of the tree, set S is partitioned, using the splitting criteria, into two samples S_1 and S_2. The sample sets S_1 and S_2 are used for the left and right subtrees of node i, respectively. The class frequencies for the interval boundaries are updated while reading the data for splitting a given node. The frequency vectors corresponding to S_1 and S_2 are updated, which avoids a separate pass over the entire data to evaluate these frequencies.

Sampling the Splitting Points with Estimation

By an additional process, Sampling the Splitting Points with Estimation (SSE), CLOUDS improves upon the above process and searches in certain selected

intervals for the exact splitting point. In the above method, the split point is a coarse estimation in the sense that the splitter is necessarily one of interval boundaries and not one of the data points. The SSE phase proceeds further from this step and examines the individual data points within the intervals. Since the number of intervals can be many, it is not practical to search the details within every interval. The SSE attempts to eliminate some of the intervals, which may not contain the actual splitter and narrows its search to only some candidate intervals to determine the exact splitter. This is accomplished by estimating the gini value for an interval. It uses a heuristic for computing an estimate. It also makes use of some characteristics of the gini function.

- The value of the gini index along a given attribute generally increases or decreases slowly. The number of good local minima is small compared to the size of the entire data set. This is especially true for attributes along which the best splitting point is obtained.
- The minimum value of the gini index for the splitting point along the splitting attribute is significantly lower than most of the other points along the splitting attribute, as well as other attributes.

CLOUDS with the SSE divides the range of the numeric attributes and finds the minimum gini value ($gini_{min}$) as described above. Further, it estimates a lower bound $gini^{est}$ of the gini value for each interval. All the intervals with $gini^{est} \geq gini_{min}$ are eliminated to derive a list of potential candidate intervals (*alive* intervals). An *alive interval* is an interval for which $gini^{est} < gini_{min}$. For an *alive* interval along each numeric attribute, the class frequencies for the intervals are computed and the gini index is evaluated at each point in the interval to find the splitter. This requires reading the entire data set corresponding to the internal node and storing the points that belong to *alive* intervals. If all the *alive* intervals do not fit into the memory, they are written onto the disk. Another pass is required to read these *alive* intervals one at a time. If the survival ratio is small, and all the alive intervals can fit into the memory, the extra pass is not required.

Computation of Estimate $GINI^{EST}$

We describe below the algorithm to compute the $gini^{est}$. The algorithm is a hill-climbing heuristic, and begins with an initial estimate of 1. For a given interval, it finds the gradient of the gini value with respect to each class. After determining the class corresponding to the minimum gradient, the set of data points that are within the interval are assumed to be shifted to the left of the interval and the gini index is computed for this change. This is repeated for each of the classes as long as the estimate monotonically decreases. The method stops at step when the estimate is not

reduced. In a similar manner, we can shift the data points to the right of the interval and get another estimate. The smaller of the two is taken as the final estimate. The algorithm is illustrated with the help of the above example.

The estimate computed for the given interval is $[z_{i-1}, z_i)$. Assume that if z_{i-1} is considered as the splitting point, it splits n data points into two groups n_1 (to the left) and n_2 (to the right of the point). From n_1 data elements there are $p_{(i-1)j}$ in class C_j, similarly, from n_2 data elements there are $q_{(i-1)j}$ in class C_j. To determine the gradient of the gini with respect to a given class and selecting the class corresponding to the smallest gradient, we compute

$$d_j = \frac{2}{n_1 n_2}\left[(p_{(i-1)j} + q_{(i-1)j})\frac{n_1}{n} - p_{(i-1)j}\right].$$

The algorithm to compute the estimate is described below.

Estimate_Left_to_right Algorithm

C be the initial list of set of classes
set gini^{est} = 1;
do for every class j until C is empty
  Let C_j be the class with smallest value d_j
  delete C_j from C.
  compute the gini index with z_{i-1} as the splitting point but replacing $p_{(i-1)j}$ by p_{ij}. (that is assuming that there are p_{ij} data elements of j^{th} class to the left of the split.)
  If the new gini index is less than the gini^{est}
  then update the estimate.
end do.

Let us consider the attribute 'Age' for illustration and the intervals are [23, 39) and [39, 55), and calculate the estimate at 39 (left to right). There are two classes, B and G

$$d_B = -1/12 \quad \text{and} \quad d_G = 1/12.$$

Though there is one point to the left of 39 of class B, we consider it as if there are 2 data elements to its left and compute the gini index. We get the gini index as 3/24. Since it is less than the initial value 1, the gini^{est} is updated as 3/24. The next class is G, but the new gini index is not less than the current estimate and hence the method returns 3/24 as the left-to-right estimate.

The performance of the SSE method depends on the quality of the estimated lower bound of the gini index and on the fraction of the elements in the *alive* intervals, that is, the survival ratio. Since the gini index is calculated for each of the splitting points in these intervals, the computational cost is proportional to the survival ratio.

6.17 BOAT

BOAT (**B**ootstrap **O**ptimistic **A**lgorithm for **T**ree Construction) is another approximate algorithm based on sampling. As the name suggests, it is based on a well-known statistical principle called *bootstraping*.

We take a very large sample D from the training data set T, so that D fits into the main memory. Now, using the bootstraping technique, we take many small samples with replacement of D as $D_1, D_2, \ldots, D_k$ and construct, by any of the known methods, decision trees $DT_1, DT_2, \ldots, DT_k$, respectively, for each of these samples. From these trees, called *bootstrap trees*, we construct a single decision tree for the sample data set D. We call this tree the sample tree. The sample tree is a coarse decision tree in the sense that the splitting conditions at any node is not of the form $A \leq n$, but it is specified in the form of a confidence interval $[z_L, z_R]$ and the exact split point n lies in this interval. After constructing the sample tree, there is a separate process, a clean-up process, when the exact value of n is determined within the confidence interval.

Construction of Sample Tree

We assume that by some standard construction method, k bootstrap trees $DT_1, DT_2, \ldots, DT_k$ are constructed from the training samples $D_1, D_2, \ldots, D_k$, respectively. In order to construct the sample tree, we process these trees in a top- down manner. For each node n, we check whether the k bootstrap splitting attributes at n are identical. If the splitting attribute is not the same for all the k bootstrap trees, then we delet node n and its subtree in all bootstrap trees.

If we have the same splitting attribute for all bootstrap trees at node n, and if the attribute is a categorical one, then we check whether the splitting criteria is same for all the bootstrap trees. If not, we delete node n and its subtree from all trees. We assume the splitting attribute for the sample tree to be the bootstrap splitting attribute, and the splitting criterion of the sample tree as the bootstrap splitting criterion.

If the bootstrap splitting attribute at node n is numerical, we consider all the k bootstrap splitting points to define a confidence interval $[z_L, z_R]$. This confidence interval becomes the coarse splitting criterion for the sample tree. The exact splitting point is assumed to lie within this confidence interval. The method of computing the exact splitting point is described below.

Computation of Exact Splitting Criteria

We assume that the sample tree is available with coarse splitting conditions for each of the numerical attributes. The algorithm is based on the assumption that the bootstraping technique results in correct coarse splitting conditions. That is, the exact

splitting point lies within the confidence interval so computed, with high probability. We make one scan over the database to determine the class histogram for the portion of the original training set that lies within the confidence intervals. Then, the splitting indices are computed at each distinct value of the attributes within the confidence interval. This process can be achieved in just one pass for the whole tree.

The confidence interval can be strengthened by constructing a larger number of bootstrap trees. For certain data sets, if the original sample D is not a proper representative, the sample tree may be very small. This is because we may be deleting the portions of the tree which are not in consensus in all bootstrap trees. It may happen that we delete a major portion of the bootstrap trees. In such cases, we should take one more sample D and start constructing afresh.

6.18 Pruning Technique

The decision tree built using the training set, deals correctly with most of the records in the training set. This is inherent to the way it was built. Overfitting is one of unavoidable situations that may arise due to such construction. Moreover, if the tree becomes very deep, lopsided or bushy, the rules yielding from the trees become unmanageable and difficult to comprehend. Therefore, a pruning phase is invoked after the construction to arrive at a (sort of) optimal decision tree.

The pruning of the decision tree is done by replacing a subtree by a leaf node. The replacement takes place if a decision rule establishes that the expected error rate in the subtree is greater than in the single leaf.

There are two schools of thought in adopting pruning strategies. They are based on whether to use the same training data set that is used for building the tree for pruning, or to use a separate test data set for pruning. Both the training data set and test data set are pre-classified data, that is, the class labels of each tuple are known for this set. Most often, it is hard to have a large collection of such data to warrant both the test and training sets. One may think of using the complete set of data for training. In that case, pruning is carried out based on the training set. In case the known data is sufficiently large so as to keep aside a portion of it for testing, the second approach is preferable. CART uses the train-and-test approach for pruning. In the train-and-test approach, we train on one set and test on another—the *reduced error pruning* algorithm greedily prunes nodes that gain information on the test set. The error is measured based on test data classification. The *post-pruning approach* prunes after the rules are obtained from the tree. It removes the preconditions that improve estimated accuracy and filters out poor rules.

We concentrate on a pruning strategy, based on minimum description length.

The MDL (**M**inimum **D**escription **L**ength) principle states that the 'best' tree is the one that can be encoded using the fewest number of bits. Thus, the challenge for the

pruning phase is to find the subtree that can be encoded with the least number of bits. So it is necessary to study the scheme for encoding the decision tree.

Cost of Encoding Tree

The cost of encoding the tree comprises of three separate costs.

A. The cost of encoding the structure of the tree

The structure of the tree can be encoded by using a single bit, in order to specify whether a node of the tree is an internal node or a leaf node. A non-leaf node can be represented by 1 and a leaf node by 0.

B. The cost of encoding for each split

The cost of encoding each split involves specifying the attribute that is used to split the node and the value for the attribute. The splitting attribute can be encoded using *log a* bits, if the total number of attributes is a. The splitting point can be specified in a different way for categorical and numerical attributes. If the splitting attribute is numeric with v distinct values then, since there are $(v-1)$ possible splitting points, $\log(v-1)$ bits are needed to encode the split point. On the other hand, for a categorical attribute, there are 2^v different subsets of values, of which the empty set and the full set cannot be splitting sets. Thus, the cost of encoding the split is $\log(2^v-2)$. $C_{split}(n)$ denotes the cost of splitting the node n.

C. The cost of encoding the classes of data records in each leaf of the tree

For the cost of encoding the classes of data records T, the following equation can also be used.

$$C(T) = \sum_i n_i \log\frac{n}{n_i} + \frac{k-1}{2}\log\frac{n}{2} + \log\frac{\pi^{k/2}}{\Gamma\left(\frac{k}{2}\right)}$$

where k is the number of classes; n_i is the number records of T in class i and n is the total number of records in T.

In the above equation, the first term is simply n times the entropy of T. Also, since k never exceeds n_i the sum of the last two terms is always non-negative.

Minimum cost subtree rooted at a node

Let S be the set of records associated with a node n. If n is a leaf node, then the minimum cost subtree rooted at n is simply n itself. The cost of the cheapest subtree rooted at n is given by

$$\text{min\_cost}(n) = C(S) + 1.$$

On the other hand, if n is an internal node in the tree with children $n1$ and $n2$, then the minimum cost subtree is either the node n itself or the node n along with the children n_1 and n_2 and the minimum cost subtrees rooted at n_1 and n_2. The cost of the cheapest subtree is given by

$$\text{min\_cost}(n) = MIN[C(S)+1,\ C_{split}(n)+1+\text{min\_cost}(n_1)+\text{min\_cost}(n_2)],$$

where $C_{\text{split}}(n)$ is the cost of splitting node n and min_cost(n_i) is the minimum cost tree at n_i.

Pruning Algorithm

Now that we have formulated the cost of a tree, we next turn our attention to computing the minimum cost subtree of the tree constructed in the building phase. The main idea is that if minimum cost at a node is the cost of the node, then the splitting of the node would result in a higher cost and hence the subtree at this node is removed. In other words,

If min_cost(n) = $C(S)$ +1, then prune the child nodes n_1 and n_2.

Example 6.7

Let us consider the example discussed earlier (Figure 6.2), presented here again for ready reference.

The cost of each leaf node can be computed as follows:

Since each leaf node represents a pure node, that is, all the records at this node are in one class, the first two terms of the cost vanish.

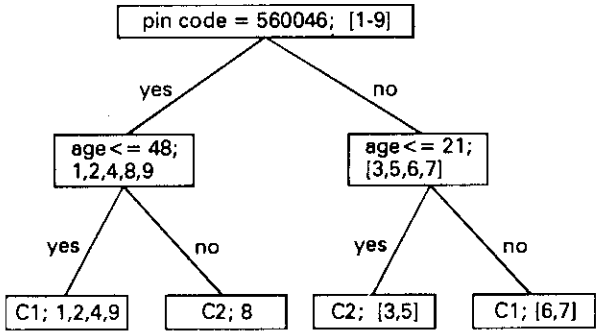


Figure 6.2

$$C(N4) = C(N5) = C(N6) = C(N7) = \log\frac{\pi^{1/2}}{\Gamma(1/2)} = 0$$

The cost of nodes at the intermediate level is given by

$$C(N2) = 4\log\frac{5}{4} + \log\frac{5}{1} + \frac{1}{2}\log\frac{5}{2} + \log\frac{\pi}{\Gamma(1)} = 5.92$$

$$C(N3) = 2\log 2 + 2\log 2 + \frac{1}{2}\log\frac{4}{2} + \log\frac{\pi}{\Gamma(1)} = 6.1522$$

$C_{split}(N2) = \log 4 = 2$ (Note that the splitting attribute at N2 is numerical)

$$\text{min\_cost}(N2) = Min[C(N2)+1, C_{split}(N2) + C(N4) + C(N5) + 1]$$
$$= Min[6.92, 3] = 3$$

So $N2$ need not be pruned.

$$\text{min\_cost}(N3) = Min[C(N3)+1,\ C_{split}(N3)+1] = Min[7.1522, 2.58] = 2.58.$$

So the node $N3$ will not be pruned.

$$C(N1) = 6\log\frac{6}{9} + 3\log\frac{3}{9} + \frac{1}{2}\log\frac{9}{2} + \log\frac{\pi}{\Gamma(1)} = 17.003$$

$C_{split}(N1) = \log(2^2 - 2) = \log 2 = 1$ (The splitting attribute is categorical)

$$\text{min\_cost}(N1) = Min[C(N1)+1,\ C_{split}(N1) + \text{min\_cost}(N2) + \text{min\_cost}(N3) + 1]$$
$$= Min[18.003, 1+3+2.58+1].$$

Hence, the node N1 is also not pruned.

6.19 INTEGRATION OF PRUNING AND CONSTRUCTION

A natural question that arises is whether we can incorporate the pruning step within the construction process of the decision tree. In other words, instead of pruning being a post-processing step after the tree is fully built, whether it is possible to identify a node that need not be expanded right at the time of construction as it is likely to be pruned eventually. We have seen that CHAID attempts to accomplish this. Recently, Rastogi and Shim have proposed a new algorithm, PUBLIC, which also attempts to integrate the pruning step within the tree-building process.

PUBLIC

Recall that the pruning method identifies whether the minimum cost subtree at that node is due to the subtree by splitting of the node or due to the node itself. During the

construction process, before expanding the node, if we can check this condition, we can decide whether to expand the node or not. The critical part is that the min_cost is calculated recursively in a bottom-up approach. The cost of a min_cost of a node cannot be known unless the cost of its subtrees are known. This becomes the main reason to employ the pruning step after the tree is fully constructed. PUBLIC (**Pru**ning and **Bu**ilding **I**ntegrated in **C**lassification) attempts to get over this limitation by estimating a lower bound of the min_cost of any node, before the node is expanded. It can easily be seen that if the pruning criterion is satisfied with the lower bound, then it is also satisfied with the actual value.

PUBLIC distinguishes among three types of current leaf nodes:

- The first kind of nodes are those that are yet to be expanded. For such nodes, the lower bound on cost is estimated, as the exact value cannot be computed.
- The second is a node which is a leaf node due to pruning its subtrees.
- The third is a node that cannot be expanded further in the construction phase. For the last two categories of nodes, the min_cost of the node can be accurately computed as $C(n)+1$.

Thus, the main part of the PUBLIC algorithm is to estimate the lower bound on the cost of the first kind of nodes. A simple estimate of the lower bound on the cost of a node is 1. We give below another algorithm to compute the lower bound.

Let us assume that for a node n, S is the set of records associated with this node and let k be the number of classes for the records in S. Let n_i be the number of records belonging to class i in S. Let us also assume, without any loss of generality, that the n_i values are arranged in decreasing order. The lower bound is estimated iteratively, with

Lower Bound Estimation

Initialize $s = 1$;

$lower\_bound = 3 + \log a + \sum_{i=3}^{k} n_i$, where a is the total number of attributes

while $s + 1 < k$ and $n_{s+1} > 2 + \log a$

    $lower\_bound = lower\_bound + 2 + \log a - n_{s+2}$

increment s;

the initial estimate being set to 1.

Note that if there are only two classes, the lower bound is directly estimated, without any iteration, as $3 + \log a$, where a is the total number of attributes.

Thus, the lower bound so computed is compared with the cost of the node and a decision is taken whether to stop expanding the node at this stage.

6.20 Summary: An Ideal Algorithm

After studying different strategies for building a decision tree for a large database, let us try to build an ideal decision tree algorithm based on these principles.

We are given a training data set. First of all, we must discover the concept hierarchy of each of the attributes. This technique is essentially similar to dimension modelling, which we learned about in Chapter 2. This study helps in removing certain attributes from the database. As we move up this hierarchy, some of the numerical attributes may turn into categorical ones. We now identify the numerical and categorical attributes. Just having the numerical data type does not necessarily imply that the attribute is numerical (for example, *pincode*). We take a large sample of the training set so that the sample fits into the memory. Sampling involves *select* operations (tuple sampling) and *project* operations (attribute removal). Construct the attribute list for each attribute. Construct a generalized attribute list for each attribute, based on dimension hierarchy (how do you do this? Given as exercise). Construct a coarse decision tree. It is coarse because of many reasons: (a) generalized attribute list, (b) bootstraping, and (c) discretizing on the sample data. Search for the exact splitting point from the coarse splitting point. Prune the tree using the MDL scheme. The question is whether to first prune the coarse tree and then search for exact splitting point or whether the exact tree to be pruned can be investigated. Generate rules from the tree and simplify the rules if a test data set is available.

There are many other strategies proposed in the literature. For example, clustering techniques can be used for determining splitting attributes (for example, SPEC). Oblique trees can also be an useful strategy. Some researchers formulate the decision tree problem also as an optimization problem.

6.21 Other Topics

We shall focus on some of the related topics here. Though these topics do not directly contribute to the study of the construction process, they are certainly helpful while applying the decision tree process for data mining.

Parallel Algorithms

The decision tree construction algorithms have some natural characteristics that prompts for parallclization. Once a node is generated, all of its children can be simultaneously expanded. Furthermore, the process of computing the splitting attributes and criteria can also be decomposed by performing data decomposition on the training data set. Nevertheless, parallelization of the decision tree construction algorithm is a challenging problem for the reasons given below:

- The shape of the decision tree is very irregular and cannot be predicted beforehand.
- The amount of work associated with each node varies and is data-dependent. Hence, any static allocation scheme may not work well with regard to proper load balancing.
- There can be high-cost data movement. This is because the training set available at any node is to be partitioned and distributed over all its child nodes for concurrent processing.

The general approach of parallel algorithms for constructing a decision tree are

- node-based decomposition, and
- attribute-based decomposition.

SPRINT, and ScalParc are some parallel decision tree construction methods.

Oblique Trees

It is not always possible to have data sets in real-life such that all the attributes are free of any correlations. If two attributes are highly correlated in any application, the decision tree with a single attribute-splitting criteria is not useful. In such cases, construct trees with a splitting criteria involving more than one attribute at any node.

Incremental Computation

Very often, we come across dynamic situations where the database is constantly incremented. It is necessary to devise a decision tree algorithm that can be updated with incremental training data sets. BOAT can support the incremental induction of a decision tree. However, not many researchers concentrate on such methods. One argument could be that in a supervised training environment, training may be an expensive process and should not be carried out very frequently. As the database is incremented, only the classification phase can be carried out to update the error of classification.

6.22 Conclusion

Classification is one of the important problems in data mining, and particularly classification using decision trees is currently one of the most active research areas in data mining research. We have tried to give a detailed account of this subject in this chapter. It will be of great help for a student to learn the techniques of constructing decision trees for possible application in data mining. It is envisaged that this chapter will also introduce researcher to state-of-art work on this topic.

Further Reading

Decision trees are a very well investigated area and a large number of results have been documented on diverse research areas. The foundations of Classification Tree and Decision Tree are dealt with in [Breimann, 1984], [Quinlan, 1986]. Quinlan proposed ID3 in [Quinlan, 1986] and C4.5 in [Quinlan, 1993]. Magidson proposed CHAID in 1993 [Magidson, 193]. SLIQ was introduced by [Mehta, 1996] and subsequent improvement as SPRINT was proposed by [Shafer, 1996]. Readers are referred to [Efron, 1993] for fundamentals of the bootstrap techniques. The BOAT algorithm was proposed by Gehrke, Ganti, Ramakrishnan and Loh in [Gehrke, 1999]. Rastogi *et al.* described PUBLIC in [Rastogi, 1998]. The RainForest family of algorithms can be found in [Gehrke, 1998]. CLOUDS was introduced in [Alsabti, 1998]. ScalParC was given in [Joshi, 1998]. The MDL-based pruning method was given in [Mehta, 1995]. Oblique decision tree concepts are dealt in [Fukuda, 1996] and [Murthy, 1994]. The parallel implementation of the decision tree was proposed by Vipin Kumar and his team in [Srivastava, 1997]. Different pruning methods are compared in [Mingers, 1989].

Exercises

1. Describe the essential features in a decision tree. How is it useful to classify data?
2. What is a classification problem? What is supervised classification? How is a decision tree useful in classification?
3. What is a splitting attribute? What is a splitting criterion? Are the splitting criteria different for numeric attributes and categorical attributes?
4. What are the different methods of computing the best split? What is the gini index? What are entropy gain and gain ratio?
5. What are the disadvantages of the decision tree over other classification techniques?
6. What are advantages and disadvantages of the decision tree approach over other approaches of data mining?
7. Describe the ID3 algorithm of the decision tree construction. Why is it unsuitable for data mining applications?
8. Describe the class histogram, count matrix and AVC-sets. Are they similar in some respect?
9. Discuss the SLIQ and SPRINT algorithms. How do they differ from each other? Why are they suitable for data mining? Compare these algorithms with ID3 and CART.
10. What is windowing? When would you prefer to use this technique?
11. Describe the PUBLIC algorithm. How does it handle pruning while building the tree?

12. Describe the estimation of gini index for a node used in the PUBLIC algorithm.
13. Describe the CLOUD algorithm and discuss its advantages over SLIQ.
14. Describe the RainForest algorithm and compare its features with other algorithms.
15. Based on the algorithms described in the text, devise an ideal decision tree algorithm and give reasons for selecting the different strategies.
16. Let us assume that the training data set has attributes with three levels of dimension hierarchy. How would you maintain the count matrix for different levels of this hierarchy?
17. Describe different pruning strategies.
18. What is Attribute Induction? What is Relevance Analysis? In what context are these useful?
19. What are the three phases of construction of a decision tree? Describe the importance of each of the phases.
20. Describe the generic algorithm of the decision tree construction method.
21. Decision tree classification is a supervised classification because in addition to classifying the data set, it also generates the descriptions of the classes. True or False?
22. Overfitting is an inherent characteristics of decision tree and its occurrence depends on the construction process and not on the training data set. True or False?
23. Pruning is essentially to avoid overfitting. True or False?
24. The accuracy of the classification of a decision tree is calculated using the gini index. True or False?
25. Bootstrapping is carried out in the main memory. True or False?
26. The coarse numeric splitting criterion is a range of values, whereas there is nothing like a coarse splitting criterion for categorical attributes. True or False?
27. An oblique tree is relevant when
    a. The attributes are correlated
    b. The attributes are independent
    c. There are only two attributes
    d. All attributes are categorical
28. The ID3 generates a
    a. Binary decision tree
    b. A decision tree with as many branches as there are distinct values of the attribute
    c. A tree with a variable number of branches, not related to the domain of the attributes
    d. A tree with an exponential number of branches.
29. Gain ratio has an advantage over Gain when an attribute is
    a. Categorical with only two possible values
    b. Numerical with two possible values
    c. Categorical with a large number of distinct values
    d. Numerical with a large number of distinct values

30. In order to select the attribute to split at a given node by taking the gini index, we select the attribute which has the
    a. Maximum value of the gini index
    b. Minimum value of the gini index
31. The importance of using an attribute list is
    a. To avoid repeated sorting of the data set
    b. To get a more accurate splitting criterion
    c. To have an alternative method for the gini index and gain index
    d. To approximate the data set to a smaller size.
32. Which of the following algorithms integrates the construction and the pruning phase?
    a. CLOUD
    b. RainForest
    c. PUBLIC
    d. CART
33. Which of the following is an approximate algorithm?
    a. ID3
    b. C4.5
    c. SLIQ
    d. CLOUD
34. The count matrix is used in
    a. PUBLIC algorithm for categorical attributes
    b. SPRINT/SLIQ algorithms for numeric attributes
    c. CLOUD algorithm for numeric attributes
35. The following example represents the training database that gives information of whether or not to play golf, given a set of climatic conditions. Illustrate the working of PUBLIC, CLOUD and SPRINT using this example.

Outlook	Temp	Humidity	Windy	Class
sunny	75	70	true	play
sunny	80	90	true	no play
sunny	85	85	false	no play
sunny	72	95	false	no play
sunny	69	70	false	play
overcast	72	90	true	play
overcast	83	78	false	play
overcast	64	65	true	play
rainy	81	75	false	play
rainy	71	80	true	no play
rainy	65	70	true	no play
rainy	75	80	false	play
rainy	68	80	false	play

36. Consider the following examples

MOTOR	WHEEELS	DOORS	SIZE	TYPE	CLASS
NO	2	0	small	cycle	bicycle
NO	3	0	small	cycle	tricycle
YES	2	0	small	cycle	motorcycle
YES	4	2	small	automobile	Sports car
YES	4	3	medium	automobile	minivan
YES	4	4	medium	automobile	sedan
YES	4	4	large	automobile	sumo

Use this example to illustrate the working of different algorithms.

BIBLIOGRAPHY

Alsabti Khaled, Ranka Sanjay, and Singh Vineet. CLOUDS: A decision tree classifier for large datasets. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD-98)*, New York City, August 1998.

Bioch Jan C., van der Meer Onno, and Potharst Rob. Invariate Decision Trees. In *Principles of Data Mining and Knowledge Discovery*, First European Symposium, PKDD'97, Komorowski and Zytkow (eds.), Springer LNAI 1263, pp. 232–242, Trondheim, Norway, 1997.

Breiman, Friedman, and Olshen. *Stone: Classification and Decision Trees.* Wadsworth, 1984.

Cohen W. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning,* Lake Tahoe, California, 1995.

Corruble V., Brown D.E., and Pittard C.L. A comparison of decision classifiers with backpropagation neural network for multimodal classification problems. *Pattern Recognition*, **26**: 953–961, 1993.

Efron B., and Tibshirani R.J. *An Introduction to the Bootstrap.* Chapman & Hall, 1993.

Fayyad U.M. On the induction of decision trees for multiple concept learning. PhD Thesis, University of Michigan, 1991.

Frank E., and Witten I.H. Generating accurate rule sets without global optimization. In *Proceedings of the International Conference on Machine Learning*, 1998.

Fukuda T., Morimoto Y., Morishita S., and Tokuyama T. Data mining using two-dimensional optimized association rule. *SIGMOD* 1996.

Frank E., Wang Y., Inglis S., Holmes, G., and Witten. I.H. Using model trees for classification. *Machine Learning* **32** (1), 1998, pp. 63–76.

Gehrke J.E., Ramakrishnan R., and Ganti V. RAINFOREST: A framework for fast decision tree construction of large data sets. In *Proceedings of the 24th International Conference on Very Large Databases*, New York, 1998.

Gehrke J.E., Ganti V., Ramakrishnan R., and Loh W-.Y. BOAT—Optimistic decision tree construction. In *Proceedings of SIGMOD '99,* Philadelphia, 1999.

Joshi Mahesh V., Karypis George, and Kumar Vipin. ScalParC: A new scalable and efficient parallel classification algorithm for mining large datasets. In *Proceedings of the 12th International Parallel Processing Symposium (IPPS/SPDP),* Orlando, USA, April 1998.

Kamber M., Winstone L., Gong W., Cheng S., and Han J. Generalization and decision tree induction: efficient classification in data mining.*Technical Report*, Simon Fraser University.

Liu Bing, and Hsu Wynne. Post analysis of learned rules. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, Portland, Oregon, USA, Aug 4-8, 1996, pp. 828-834.

Liu Bing, Hsu Wynne, and Chen Shu. Discovering conforming and unexpected classification rules. IJCAI-97 In *Workshop on Intelligent Data Analysis in Medicine and Pharmacology (IDAMAP-97)*, Nagoya, Japan, August 23–29, 1997.

Liu Bing, Hsu Wynne, and Chen Shu. Using general impressions to analyze discovered classification rules. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (KDD-97)*, Newport Beach, California, USA, pp. 31–36, August 14–17, 1997.

Liu Bing, Hsu Wynne, and Ma Yiming. Integrating classification and association rule mining. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD-98)*, New York, USA, 1998.

Magidson J. The CHAID approach to segmentation modelling. In *Bagozzi,R.T. (ed.) Advanced Methods of Marketing Research*, Blackwell Publishing, Cambridge, Mass., USA, 1995.

Mehta M. Rissanen J., and Agrawal R. MDL-based decision tree pruning. In *Proceedings of the 1st International Conference on Knowledge Discovery in Databases and Data Mining*, Montreal, Canada, August 1995.

Mehta M., Agrawal R., and Rissanen J. SLIQ: A fast scalable classifier for data mining. In *Proceedings of the 5th International Conference on Extending Database*

Mingers J. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, **4**:3 227–243, 1989.

Murthy S.K., Kasif S., Salzberg S., and Beigel R. OC1: Randomized induction of oblique decision trees. In *Proceedings of AAAI'93*, 1993.

Murthy S.K., Kasif S., and Salzberg S. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research* **2**:1–32, 1994.

Quinlan J.R. Induction of decision tree. *Machine Learning*, **1**:81–106, 1986.

Quinlan J.R. C4.5: *Programs for Machine Learning*. Morgan Kauffman, 1993.

Rachlin J., Kasif S., Salzberg S., and Aha D. Towards a better understanding of memory-based and Bayesian classifiers. In *Proceedings 1994 International Conference on Machine Learning*, New Brunswick, NJ, 1994, pp. 242–250.

Rastogi Rajeev and Shim Kyuseok. PUBLIC: A decision tree classifier that integrates building and pruning. In *Proceedings of the 24th International Conference on Very Large Databases* (VLDB98), New York, 1998.

Shafer J.C., Agrawal R., and Mehta. M. SPRINT: A scalable parallel classifier for data mining. In *Proceedings of the 22nd International Conference on Very Large Databases*, Mumbai (Bombay), India, September 1996.

Srivastava A., Han E-. H., Kumar V. and Singh V. Parallel Formulation of Decision Tree classification algorithms. *Technical Report*, University of Minnesota, 1997.

Wang K., and Liu. B. Concurrent discretization of multiple attributes. *PRICAI 98*, Singapore, August 1998.

Zaki M.J., Ho C-T., and Agrawal R. Scalable parallel classification for data mining on shared-memory multiprocessors. In *IEEE International Conference on Data Engineering*, March 1999.

7

OTHER TECHNIQUES

7.1 INTRODUCTION

From Chapter 4 through to Chapter 6, we have studied in detail the algorithms for discovering association rules, for clustering and for building decision trees. In fact, these are three main areas of research in data mining algorithms in recent years. Data mining is essentially a task of learning from data and hence, any known technique which attempts to learn from data can, in principle, be applied for data mining purposes. But it is crucial, as we have noticed in the earlier chapters, that these algorithms should be suitably modified to handle the large data residing in secondary memory. This problem is better illustrated by considering the case of propositional reasoning. Since the days of Boole to the current state in machine learning and propositional reasoning of AI, researchers have been proposing algorithms with increasing efficiency, for computing prime implicates/ implicants, converting a CNF to DNF, and so on. It is a trivial exercise to notice that these problems are directly related to finding frequent sets. Readers may refer to Heiki Mannila's pioneering work in this direction. But for the theoretical importance, these algorithms are inefficient to handle disk-resident data and unless appropriately modified, these are unsuitable in practice for data mining applications. This is just a single case of

a variety of known techniques in other disciplines which share same theoretical foundation as the techniques required for data mining, but cannot be readily applied for practical data mining applications. We have noted the tricks of modifying decision trees and clustering algorithms in Chapter 5 and 6. In general, data mining algorithms aim at minimizing I/O operations of disk-resident data, whereas conventional algorithmics are more concerned about time and space complexities, accuracy and convergence. Besides the techniques discussed in the earlier chapters, a few other techniques hold promise of being suitable for data mining purposes. These are Neural Networks (NN), Genetic Algorithms(GA), Rough Set Theory and Support Vector Machines (SVM). There are many books available on these topics. The intention of this chapter is to briefly present the underlying concepts of these subjects and demonstrate their applicability to data mining. We envisage that in the coming years these techniques are going to be important areas of data mining techniques.

In Section 7.2, we introduce the principles of neural computing; and the outline of MLP architecture and RBF architecture are given. Section 7.3 discusses the learning techniques of MLP and RBF. Section 7.4 deals with one of the most popular techniques namely, Kohonen's model of Self-organizing Maps. We present a case study of data mining using NN techniques in Section 7.5.

In Section 7.6 we shall study the underlying principles of Genetic Algorithms (GA). In Section 7.9, a case study of data mining using GA is presented.

Section 7.10 gives a brief account of the Rough Set Theory. Rough sets are increasingly becoming popular in the data mining community. In the following section, we discuss a case study.

Section 7.12 is concerned with Support Vector Machines(SVM). A few techniques of data mining using SVMs are proposed.

7.2 What is a Neural Network?

Neural networks are a different paradigm for computing, which draws its inspiration from neuroscience. The human brain consists of a network of neurons, each of which is made up of a number of nerve fibres called *dendrites,* connected to the *cell body* where the *cell nucleus* is located. The *axon* is a long, single fibre that originates from the cell body and branches near its end into a number of *strands*. At the ends of these strands are the transmitting ends of the *synapses* that connect to other biological neurons through the receiving ends of the synapses found on the dendrites as well as the cell body of biological neurons. A single axon typically makes thousands of synapses with other neurons. The transmission process is a complex chemical process which effectively increases or decreases the electrical potential within the cell body of the receiving neuron. When this electrical potential reaches a threshold value (action potential), it enters its excitatory state and is said to *fire*. It is the connectivity of the

neuron that give these simple 'devices' their real power.

Artificial neurons (or processing elements, PE) are highly simplified models of biological neurons. As in biological neurons, an artificial neuron has a number of inputs, a cell body (most often consisting of the summing node and the transfer function), and an output which can be connected to a number of other artificial neurons. Artificial neural networks are densely interconnected networks of PEs, together with a rule (learning rule) to adjust the strength of the connections between the units in response to externally supplied data.

The evolution of neural networks as a new computational model originates from the pioneering work of McCulloch and Pitts in 1943. They suggested a simple model of a neuron that commuted the weighted sum of the inputs to the neuron and an output of 1 or 0, according to whether the sum was over a threshold value or not. A 0 output would correspond to the inhibitory state of the neuron, while a 1 output would correspond to the excitatory state of the neuron. Consider a simple example illustrated below.

The network has 2 binary inputs, I_0 and I_1, and one binary output Y. W_0 and W_1 are the connection strengths of input 1 and input 2, respectively. Thus, the total input received at the processing unit is given by

$$W_0 I_0 + W_1 I_1 - W_b,$$

where W_b is the threshold (in another notational convention, it is viewed as the bias). The output Y takes on the value 1, if $W_0 I_0 + W_1 I_1 - W_b > 0$ and, otherwise, it is 0 if $W_0 I_0 + W_1 I_1 - W_b \leq 0$.

But the model, known as perceptron, was far from a true model of a biological neuron as, for a start, the biological neuron's output is a continuous function rather than a step function. This model also has a limited computational capability as it represents only a linear-separation. For two classes of inputs which are linearly separable, we can find the weights such that the network returns 1 as output for one class and 0 for another class.

There have been many improvements on this simple model and many architectures have been presented in recently. As a first step, the threshold function or the step function is replaced by other more general, continuous functions called *activation*

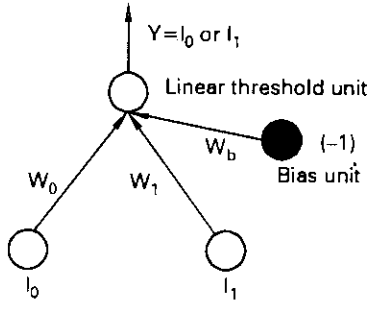


Figure 7.1 A Simple Perceptron

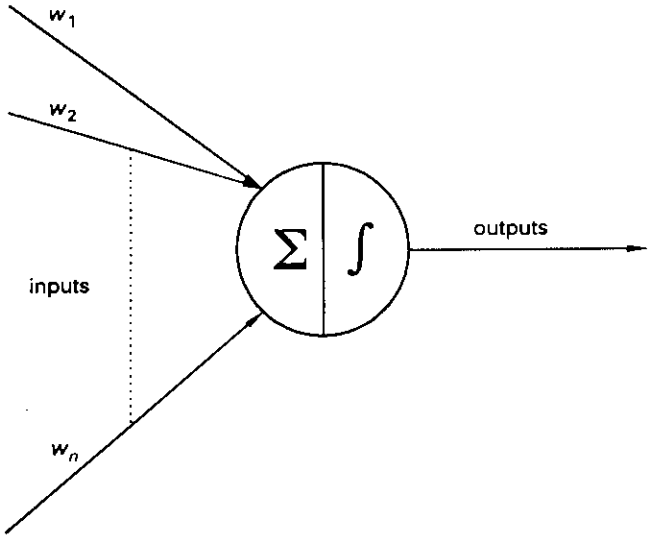


Figure 7.2 A Typical Artificial Neuron with Activation Function

functions. Figure 7.2 illustrates the structure of a node (PE) with an activation function. For this particular node, n weighted inputs (denoted W_i, $i = 1,\ldots, n$) are combined via a *combination function* that often consists of a simple summation. A *transfer function* then calculates a corresponding value, the result yielding a single output, usually between 0 and 1. Together, the combination function and the transfer function make up the *activation function* of the node.

Three common transfer functions are the sigmoid, linear and hyperbolic functions. The sigmoid function (also known as the logistic function) is very widely used and it produces values between 0 and 1 for any input from the combination function. The sigmoid function is given by (the subscript n identifies a PE):

$$Y = \frac{1}{1+\exp(-\zeta(\sum_i W_{in} X_{in}))}.$$

Note that the function is strictly positive and defined for all values of the input. When plotted, the graph takes on a sigmoid shape, with an inflection point at (0, .5) in the Cartesian plane. The graph (Figure 7.3) plots the different values of ζ as the input varies from –10 to 10.

Individual nodes are linked together in different ways to create neural networks. In a feed-forward network, the connections between layers are unidirectional from input to output. We discuss below two different architectures of the feed-forward network, Multi-Layer Perceptron and Radial-Basis Function.

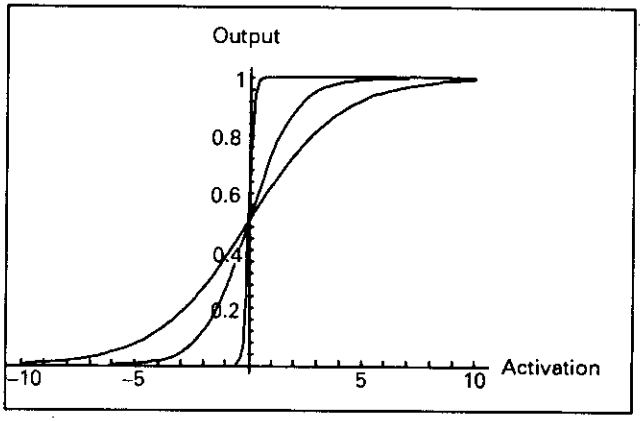


Figure 7.3 Sigmoid Functions

Multi-Layer Perceptron (MLP)

MLP is a development from the simple perceptron (Figure 7.2) in which extra *hidden layers* (layers additional to the input and output layers, not connected externally) are added. More than one hidden layer can be used. The network topology is constrained to be *feedforward,* i.e., loop-free. Generally, connections are allowed from the input layer to the first (and possibly only) hidden layer; from the first hidden layer to the

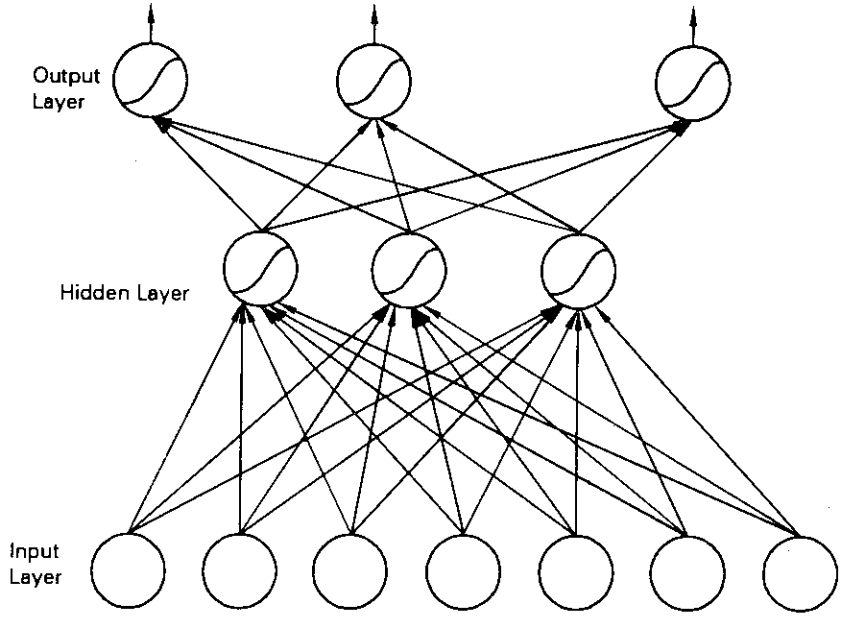


Figure 7.4 Multi-Layer Perceptron

second, and so on, until the last hidden layer to the output layer. The presence of these layers allows an ANN to approximate a variety of non-linear functions. The actual construction of a network, as well as the determination of the number of hidden layers and the determination of the overall number of units, is something of a trial-and-error process, determined by the nature of the problem at hand. The transfer function is generally a sigmoid function. The architecture is illustrated above (Figure 7.4).

Radial Basis Function Networks

Radial Basis Function (RBF) networks are also feedforward, but have only *one* hidden layer. Like MLP, RBF nets can learn arbitrary mappings; the primary difference is in the hidden layer. RBF hidden layer units have a *receptive field* which has a *centre;* that is, a particular input value at which they have a maximal output. Their output tails off as the input moves away from this point. Generally, the hidden units have a Gaussian transfer function. Figure 7.6 illustrates the Gaussian functions. In Figure 7.5 the architecture of RBF is described.

7.3 Learning in NN

In order to fit a particular ANN to a particular problem, it must be *trained* (or *learned)* to generate a correct response for a given set of inputs. *Unsupervised*

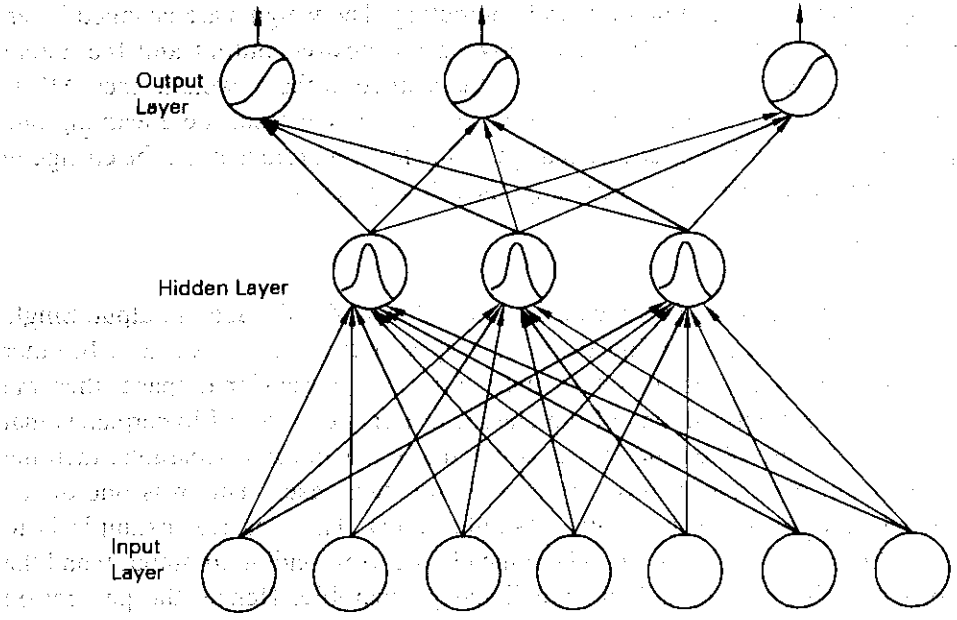


Figure 7.5 RBF Arcitecture

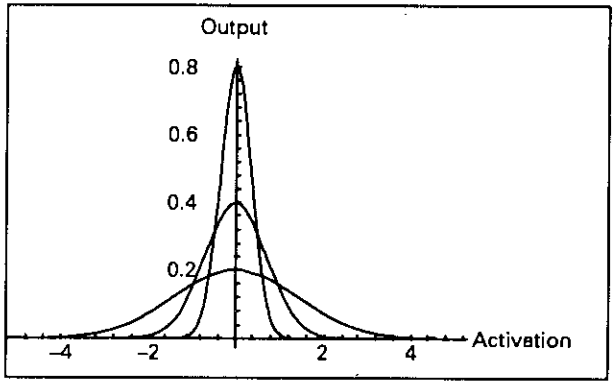


Figure 7.6 Gaussian Transfer Function. Gaussians with three different standard deviations

training may be used when a clear link between input data sets and target output values does not exist. *Supervised* training involves providing an ANN with specified input and output values, and allowing it to iteratively reach a solution. MLP and RBF employ the supervised mode of learning. We describe below these learning schemes.

Perceptron Learning Rule

This is the first learning scheme of neural computing. The weights are changed by an amount proportional to the difference between the desired output and the actual output. If W is the weight vector and ΔW_i is the change in the i^{th} weight, then ΔW_i is proportional to a term which is the error times the input. We associate a learning rate parameter to decide the magnitude of change. If the learning rate is high, the change in the weight is bigger at every step. Formally, the rule is given by

$$\Delta W_i = \xi (D - Y).I_i,$$

where ξ is the learning rate, D is the desired output, and Y is the actual output. Single perceptrons are limited to learning the simple hyperplane decision surface. In other words, if the classes are visualized geometrically in n-dimensional space, then the perceptron generates descriptions of the classes in terms of a set of hyperplanes that separate these classes. When the classes are actually not linearly separable, then the perceptron is not effective in properly classifying such cases. This was one of the major disadvantages of the perceptron (single layer). The classical example is to compute XOR by a neural network, where the class corresponding to output 0 and the class corresponding to output 1 are not linearly separable. Hence, the perceptron cannot compute the XOR function.

Training in MLP

The multi-layer perceptron overcomes the above shortcoming of the single layer perceptron. But learning in MLP is not trivial. The idea is to carry out the computation layer-wise, moving in the forward direction. Similarly, the weight adjustment can be done layer-wise, by moving in a backward direction. For the nodes in the output layer, it is easy to compute the error as we know the actual outcome and the desired result. For the nodes in the hidden layers, since we do not know the desired result, we propagate the error computed in the last layer backward. This process gives the change in the weight for the edges layer-wise. This standard method used in training MLPs is called the *back propagation* algorithm. Briefly, it involves using a gradient-descent approach to minimize the squared error between generated output values and target output values. Though there is a risk of being "trapped" in the local minima by employing such a method (convergence to a global minimum is *not* guaranteed), it has proved very efficient. The weight change rule is a development of the perceptron learning rule. The weights are changed by an amount proportional to *(the error at that unit)* times *(the output of the unit feeding into the weight)*.

Formally, the learning steps consist of the

Forward pass
The outputs and the error at the output units are calculated.

Backward pass
The output unit error is used to alter weights on the output units. Then, the error at the hidden nodes is calculated (by *back-propagating* the error at the output units through the weights), and the weights on the hidden nodes are altered using these values.

For each training data, a forward pass and backward pass is performed. This is repeated over and over again, until the error is at an acceptably low level (or we give up!).

Training RBF Networks

The RBF design involves deciding on their centres and the sharpness (standard deviation) of their Gaussians. Generally, the centres and SDs (standard deviations) are decided first by examining the vectors in the training data. RBF networks are trained in a similar way as MLP. The output layer weights are trained using the delta rule.

MLP is the most widely applied neural network technique. RBFs have the advantage that one can add extra units with their centres near parts of the input, which are difficult to classify.

7.4 Unsupervised Learning

Simple perceptrons, MLP, and RBF networks are supervised networks. In an

unsupervised mode, the network adapts purely in response to its inputs. Such networks can learn to pick out structures in their input. One of the most popular models in the unsupervised framework is the *self-organizing map* (SOM).

Competitive Learning

Competitive learning or *winner-takes-all* may be regarded as the basis of a number of unsupervised learning strategies. A competitive learning consists of k units with weight vectors w_k, of equal dimension to the input data. During the learning process, the unit with its weight vector closest (in some metric) to the input vector x is adapted in such a way that the weight vector becomes closer to the input vector after the adaptation. The unit with the closest weight vector is termed as the winner of the selection process. This learning strategy is generally implemented by gradually reducing the difference between the weight vector and input vector. The actual amount of reduction at each learning step may be guided by means of the so-called *learning rate*, ζ. During the learning process, the weight vectors converge towards the mean of the set of input data.

Kohonen's SOM

The self-organizing map (SOM) was a neural network model developed by Teuvo Kohonen during 1979-82. SOM is one of the most widely used unsupervised NN models and employs competitive learning steps. It consists of a layer of input units, each of which is fully connected to a set of output units. These output units are arranged in some topology (the most common choice is a two-dimensional grid. See Figure 7.7). The input units, after receiving the input patterns X, propagate them as they are onto the output units. Each of the output units k is assigned a weight vector w_k. During the learning step, the unit c corresponding to the highest activity level with respect to a randomly-selected input pattern X, is adapted in a such a way that it exhibits an even higher activity level at a future presentation of X. This is

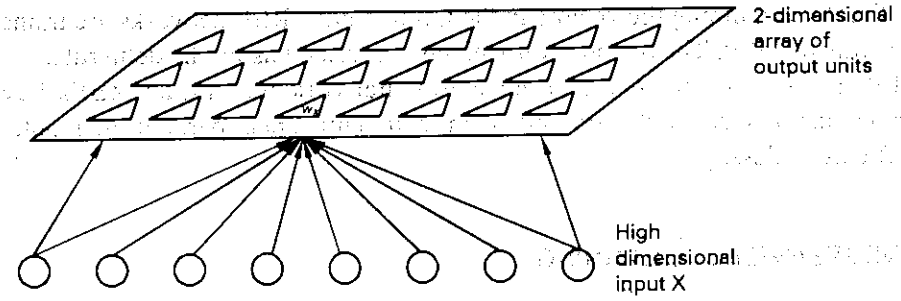


Figure 7.7 SOM architecture

accomplished by the competitive learning described above. Usually, the similarity metric is chosen to be the Euclidean distance. During the learning steps of SOM, a set of units around the winner is tuned towards the currently presented input pattern enabling a spatial arrangement of the input patterns, such that similar inputs are mapped onto regions close to each other in the grid of output units. Thus, the training process of SOM results in a topological organization of the input patterns. It is, in some sense, related to k-means clustering. Thus, SOM takes a high-dimensional input and clusters it, but still retains some topological ordering of the output. After training, an input will cause some of the output units in some area to become active. Such clustering (and dimensionality reduction) is very useful as a preprocessing stage, whether for further neural network data processing, or for more traditional techniques.

Applications of Neural Networks

These days, neural networks are used in a very large number of applications. We list here some of those relevant to our study. Neural networks are being used in

- investment analysis:
  to predict the movement of stocks, currencies etc., from previous data. There, they are replacing earlier simpler linear models.
- monitoring:
  networks have been used to monitor the state of aircraft engines. By monitoring vibration levels and sound, an early warning of engine problems can be given.
- marketing:
  neural networks have been used to improve marketing mailshots. One technique is to run a test mailshot, and look at the pattern of returns from this. The idea is to find a predictive mapping from the data known about the clients to how they have responded. This mapping is then used to direct further mailshots.

7.5 Data Mining using NN: A Case Study

In this section, we outline a case study to illustrate the potential application of NN for Data mining. This case study is taken from [Shalvi, 1996].

Knowledge Extraction through Data Mining

Kohonen, self-organizing maps (SOMs) are used to cluster a specific medical data set containing information about the patients' drugs, topographies (body locations) and morphologies (physiological abnormalities); these categories can be identified as the

three input subspaces. Data mining techniques are used to collapse the subspaces into a form suitable for network classification. The goal is to acquire medical knowledge which may lead to tool formation, automation and to assist medical decisions regarding population. The data is organized as three hierarchical trees, identified as Drugs, Topography and Morphology. The most significant portion of the morphology tree is displayed in Figure 7.8. Before presenting the data to the neural network, certain preprocessing should be done. Standard techniques can be employed to clean erroneous and redundant data.

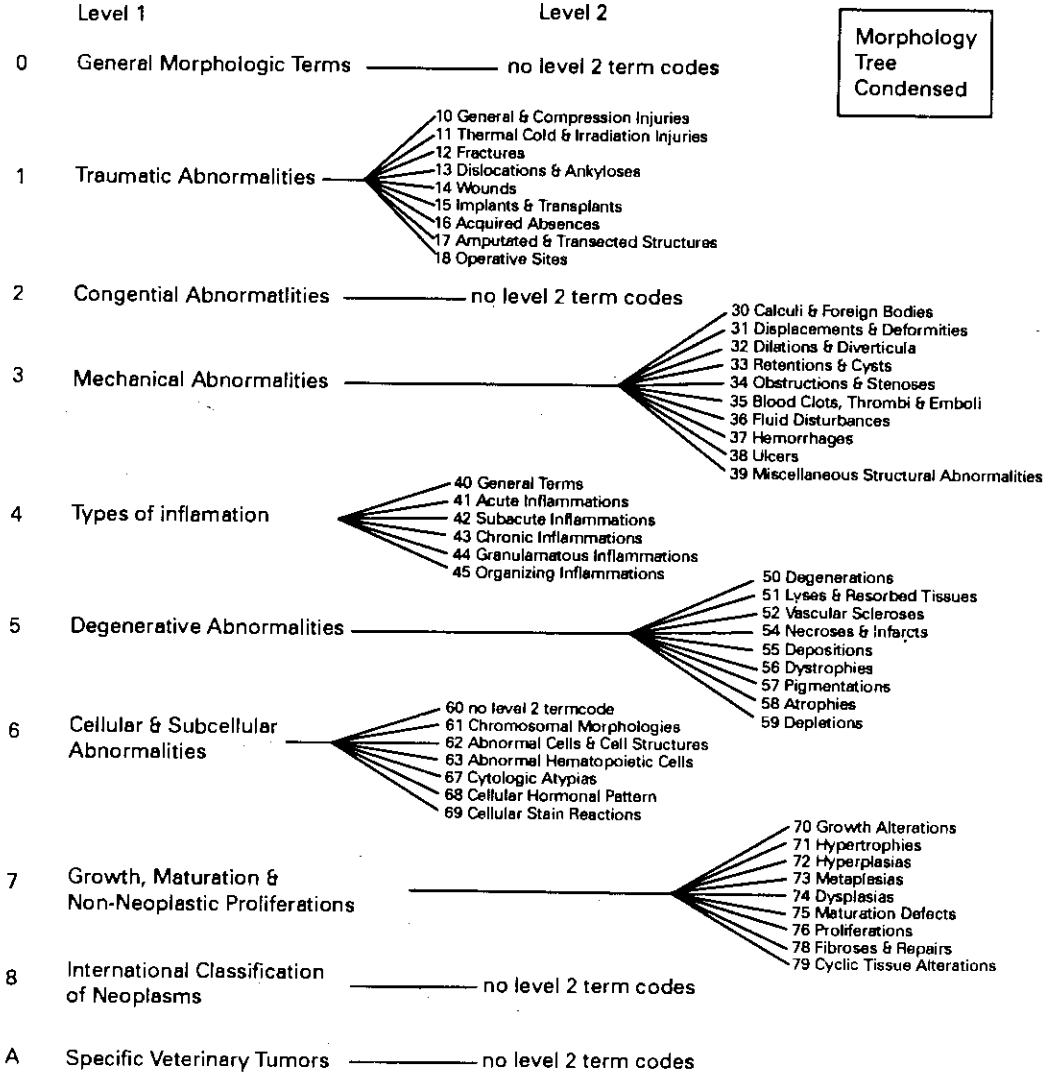


Figure 7.8 Morphology Tree

The data is processed at the root level of each tree—fourteen root level drugs, sixteen root level topographies and ten root level morphologies. By constraining all the data to the root level, the degree of differentiation has been greatly reduced from thousands to just 40. Each tuple is converted into a bipolar format. Thus, each tuple is a 40-dimensional bipolar array—a value of either 1 or –1 depending on whether any data existed for the leaves of that root node.

The Kohonen self-organizing map (SOM) was chosen to organize the data, in order to make use of a spatially ordered, 2-dimensional map of arbitrary granularity. An $n \times n$ SOM was implemented for several values of n. We describe below only the case with n=10. The input layer consists of 40 input nodes; the training set consists of 2081 tuples; and the training period was of 30 epochs. The learning coefficient ζ is initialized to 0.06. After approximately 7½ epochs, ζ is halved to 0.03. After another 7½ epochs, it is halved again to 0.015. For the final set of 7½ epochs, it is halved again to become 0.0075.

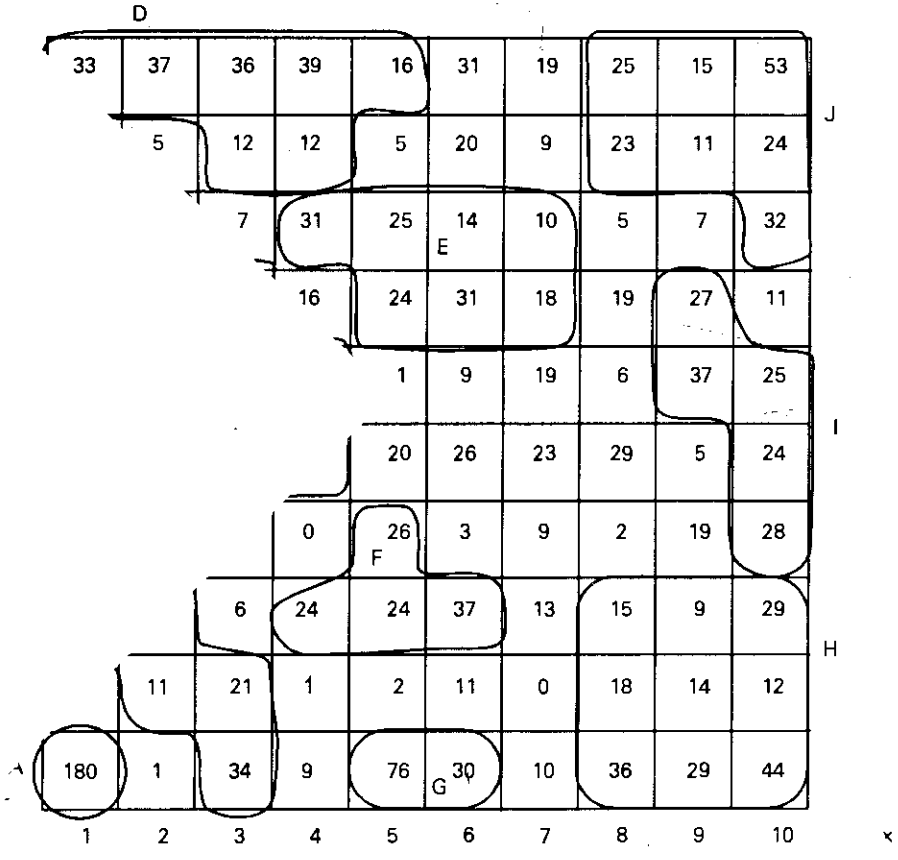


Figure 7.9 Population clusters

After the network is trained it is used for one final pass through the input data set, in which the weights are not adjusted. This provides the final classification of each input data tuple into a single node in the 10 × 10 grid. The output is taken from the coordinate layer as an (x, y) pair. The output of the SOM is a population distribution of tuples with spatial significance (Figure 7.9). This grid displays the number of tuples that were classified into each Kohonen layer node (square) during testing; for example, Square (1,1) contains 180 tuples.

Upon examination of the raw data within these clusters, one finds similarities between the tuples which are indicative of medical relationships or dependencies. Numerous hypotheses can be made regarding these relationships, many of which were not known *a priori. The* SOM groups together tuples in each square according to their similarity. The only level at which the SOM can detect similarities between tuples is at the root level of each of the three subspace trees, since this is the level of differentiation presented to the SOM's input. For example, every one of the tuples in square (1,1) contains root level data only for Drug 6, Topography 6 and Morphology 5. The tuple at square (2,1) contains these three root level nodes as well as Drug 7, a difference slight enough for the network to distinguish the tuple by classifying it one square away from (1,1). All 34 tuples in square (3,1) contain Drug 6, Topography D and Morphology 5; but only 29 of the 34 tuples contain Topography 6. Clearly, the difference between square (3,1) and square (1,1) is greater than the difference between square (2,1) and square (1,1).

7.6 Genetic Algorithm

Genetic algorithms (GA), first proposed by Holland in 1975, are a class of computational models that mimic natural evolution to solve problems in a wide variety of domains. Genetic algorithms are particularly suitable for solving complex optimization problems and for applications that require adaptive problem-solving strategies. Genetic algorithms are search algorithms based on the mechanics of natural genetics, i.e., operations existing in nature. They combine a Darwinian 'survival of the fittest' approach with a structured, yet randomized, information exchange. The advantage is that they can search complex and large amount of spaces efficiently and locate near-optimal solutions pretty rapidly. GAs were developed in the early 1970s by John Holland at the University of Michigan (*Adaptation in Natural and Artificial Systems, 1975*).

A genetic algorithm operates on a set of individual elements (the population) and there is a set of biologically inspired operators that can change these individuals. According to the evolutionary theory, only the more suited individuals in the population are likely to survive and to generate offspring, thus transmitting their biological heredity to new generations.

In computing terms, genetic algorithms map strings of numbers to each potential solution. Each solution becomes an individual in the population, and each string becomes a representation of an individual. There should be a way to derive each individual from its string representation. The genetic algorithm then manipulates the most promising strings in its search for an improved solution. The algorithm operates through a simple cycle:

- Creation of a population of strings.
- Evaluation of each string.
- Selection of the best strings.
- Genetic manipulation to create a new population of strings.

Figure 7.9 shows how these four stages interconnect. Each cycle produces a new generation of possible solutions (individuals) for a given problem. At the first stage, a population of possible solutions is created as a starting point. Each individual in this population is encoded into a string (the chromosome) to be manipulated by the genetic operators. In the next stage, the individuals are evaluated, first the individual is created from its string description (its chromosome), then its performance in relation to the target response is evaluated. This determines how fit this individual is in relation to the others in the population. Based on each individual's fitness, a selection mechanism chooses the best pairs for the genetic manipulation process. The selection policy is responsible to ensure the survival of the fittest individuals.

The manipulation process enables the genetic operators to produce a new population of individuals, the offspring, by manipulating the genetic information possessed by the pairs chosen to reproduce. This information is stored in the strings (chromosomes) that describe the individuals. Two operators are used: *Crossover* and *Mutation*. The offspring generated by this process take the place of the older

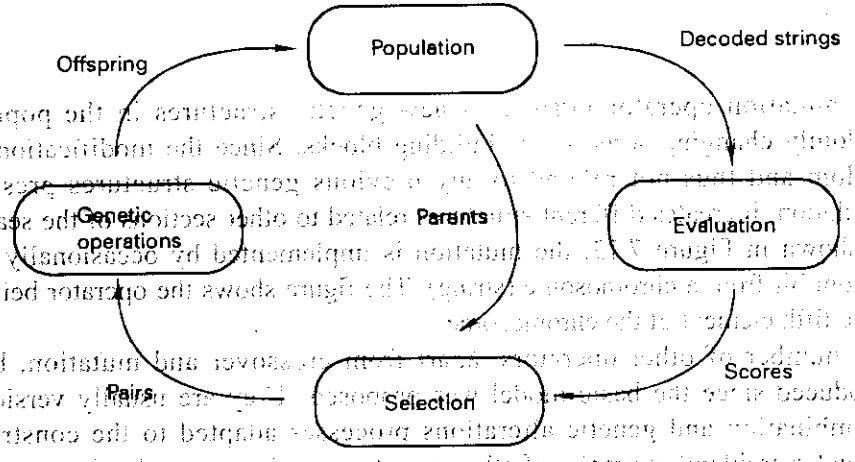
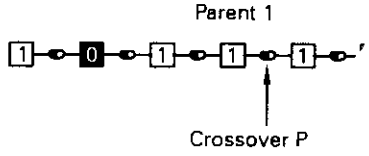


Figure 7.10 The "Reproduction" Cycle

population and the cycle is repeated until a desired lev[illegible] ained, or a determined number of cycles is reached.

Crossover

Crossover is one of the genetic operators u material. It takes two chromosomes and produce new chromosomes. As Figure randomly chosen, portions of the parer are combined to produce the new off



Figur

T
as
f

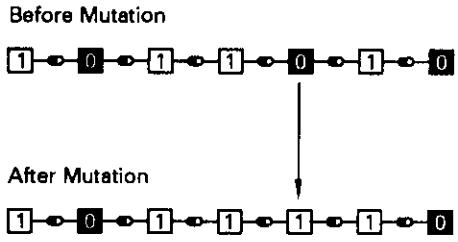


Figure 7.12 Mutation

Problem-dependent Parameters

This description of the GA's computational model reviews the steps needed to create the algorithm. However, a real implementation takes into account a number of problem-dependent parameters. For instance, the offspring produced by genetic manipulation (the next population to be evaluated) can either replace the whole population (generational approach) or just its less fit members (steady-state approach). The problem constraints will dictate the best option. Other parameters to be adjusted are the population size, crossover and mutation rates, evaluation method, and convergence criteria.

Encoding

Critical to the algorithm's performance is the choice of underlying encoding for the solution of the optimization problem (the individuals or the population). Traditionally, binary encodings have being used because they are easy to implement. The crossover and mutation operators described earlier are specific to binary encodings. When symbols other than 1 or 0 are used, the crossover and mutation operators must be tailored accordingly.

The Evaluation Step

The evaluation step in the cycle, shown in Figure 7.10, is more closely related to the actual application the algorithm is trying to optimize. It takes the strings representing the individuals of the population and, from them, creates the actual individuals to be tested. The way the individuals are coded as strings will depend on what parameters one is trying to optimize and the actual structure of possible solutions (individuals). After the actual individuals have been created, they have to be tested and scored. These two tasks again are closely related to the actual system being optimized. The testing depends on what characteristics should be optimized and the scoring. The production of a single value representing the fitness of an

individual, depends on the relative importance of each different characteristic value obtained during testing.

Data Mining using GA

The application of the genetic algorithm in the context of data mining is generally for the tasks of *hypothesis testing and refinement*, where the user poses some hypothesis and the system first evaluates the hypothesis and then seeks to refine it. Hypothesis refinement is achieved by "seeding" the system with the hypothesis and then allowing some or all parts of it to vary. One can use a variety of evaluation functions to determine the fitness of a candidate refinement. The important aspect of the GA application is the encoding of the hypothesis and the evaluation function for fitness.

Another way to use GA for data mining is to design a hybrid techniques by blending one of the known techniques with GA. For example, it is possible to use the genetic algorithm for optimal decision tree induction. As we have seen in Chapter 6, we can take samples of the training data set to build a decision tree. Thus, by randomly generating different samples, we can build many decision trees using any of the traditional techniques. But we are not sure of the optimal tree. At this stage, the GA is very useful in deciding on the optimal tree and optimal splitting attributes. The genetic algorithm evolves a population of biases for the decision tree induction algorithm. We can use a two-tiered search strategy. On the bottom tier, the traditional greedy strategy is performed through the space of the decision trees. On the top tier, one can have a genetic search in a space of biases. The attribute selection parameters are used as biases, which are used to modify the behaviour of the first tier search. In other words, the GA controls the preference for one type of decision tree over another.

An individual (a bit string) represents a bias and is evaluated by using testing data subsets. The "fitness" of the individual is the average cost of classification of the decision tree. In the next generation, the population is replaced with new individuals. The new individuals are generated from the previous generation, using mutation and crossover. The fittest individuals in the first generation have the most offspring in the second generation. After a fixed number of generations, the algorithm halts and its output is the decision tree the determined by the fittest individual. In Section 9.10, we discuss the application of GA for event prediction.

7.7 Rough Sets

Rough set theory is a tool for studying imprecision, vagueness, and uncertainty in data analysis. It focuses on delivery patterns, rules, and knowledge in data. The rough set itself is the approximation of a vague concept (set) by a pair of precise concepts, called the lower and upper approximations (which are a classification of the domain of

interest into disjoint categories). The classification (attributes) formally represents our knowledge about the domain.

Let us state the supervised classification problem in another way. Assume that our set of interest is the set S, and we know which sample elements are elements of S. We are looking for a definition of S in terms of the attributes. The membership status of objects, with respect to an arbitrary subset of the domain, may not be clearly definable. This fact leads to the definition of a set in terms of lower and upper approximations. The *lower approximation* is a description of the domain objects which are known with certainty to belong to the subset of interest, whereas the *upper approximation* is a description of the objects which may possibly belong to the subset. Any subset defined through its lower and upper approximations is called a rough set, if the boundary region is not empty. We give below the formal definition of this concept.

An equivalence relation, θ on U, is a binary relation which is transitive, reflexive and symmetric. In the context of rough sets, we term an equivalence relation as an *indiscernibility relation*. The pair (U, θ) is called an approximation space. With each equivalence relation θ, there is a partition of U such that two elements a, b in U are in the same class in this partition, iff $a\theta b$. We denote a class in the partition due to θ, as $\theta_a = \{b \in U \mid a\theta b\}$. For a subset $X \subseteq U$, we say that

$\underline{X} = \bigcup\{\theta_x \mid \theta_x \subseteq X\}$, is said to be the lower approximation or positive region of X, and

$\overline{X} = \bigcup\{\theta_x \mid x \in X\}$, is said to be the upper approximation or possible region of X.

$\overline{X} - \underline{X}$ is the area of uncertainty and the region $\underline{X} \cup \overline{X}$ is said to be the region of certainty.

The *rough set* of U is a pair $(\underline{X}, \overline{X})$.

For example, let us consider the database, T, which contains the following transactions:

$\{a\ b\ c\}$, $\{a, b, d\}$, $\{a, c, d\}$, $\{a, c, e\}$, $\{a, d, e\}$, and $\{a, d, f\}$. Assume also that the items in the transaction are ordered.

We define an equivalent relation between transactions as having two common prefixes. In other words, two transactions are equivalent if their first two elements are the same.

X is a subset of a transaction which contains transactions $\{a, b, c\}$, $\{a, b, d\}$, and $\{a, c, d\}$.

The lower approximation of X is $\{\{a, b, c\}, \{a, b, d\}\}$

The upper approximation of X is $\{\{a, b, c\}, \{a, b, d\}, \{a, c, d\}, \{a, c, e\}\}$

In the context of a database, let us assume the database, T, as a set of tuples. The set of attributes on these tuples are defined as $A=\{A_1, A_2, \ldots, A_m\}$. For a tuple $t \in T$, and for a subset $X \subseteq A$, $t[X]$ denotes the projection of the tuple t, on the set of attributes in X.

For a given subset of attributes Q, we define an equivalence relation on T as follows.

For two tuples t_1, t_2, we say that t_1 θ t_2 if $t_1[Q] = t_2[Q]$. That is, we say that two tuples t_1 and t_2 are indiscernible with respect to the attributes in Q. With this indiscernibility relation, we can determine the lower and upper approximation for any subset of tuples in T. We can say that a set of attributes Q is dependent on another set P, if the partition of the database with respect to P contains the partition with respect to Q. This will lead to efficient techniques of attribute elimination required for decision trees, association rules and clustering.

The notion of the rough set has been investigated since the 70s, and has been found useful in the realms of knowledge acquisition and data mining.

APPLICATION

A number of practical applications of this approach have been developed in recent years, in areas such as medicine, drug research, process control and others. One of the primary applications of rough sets in AI is for the purpose of knowledge analysis and discovery of data. The rough set framework has been used with success in several data mining applications.

Some of these are:

RSES: [pvv] The RSES system was developed in Poland and uses state-of-the-art techniques from the Rough Sets theory. A maximum of 30,000 objects with 16,000 attributes can be processed for rule generation. The number of attributes poses few restraints, but the relatively low number of maximal objects prevents it from being used on large data sets.

DataLogic/R It is a database "mining" from Reduct Systems Inc. The software is based on theories of knowledge representation, inductive logic and rough sets. According to Reduct Systems, their software is unique in that it analyzes logical patterns in data at different levels of knowledge representation. This means that it can discover facts and relationships not accessible with any other method.

KDD-R This system is based on the **V**ariable **P**recision **R**ough **S**et (VPRS) model. This carries out: (i) analysis of dependencies among attributes and elimination of superfluous attributes, and (ii) computation of rules from data.

LERS **L**earning from **E**xamples based on **R**ough **S**ets, is another system created for rule induction. The system handles inconsistencies in data sets by following the principles of rough sets. These inconsistencies are not corrected, but instead the upper and lower approximation for each concept is calculated. Thereafter, deterministic and indeterministic rules are generated.

7.8 Support Vector Machines

Support vector machines are learning machines that can perform binary classification and regression estimation tasks. They are becoming increasingly popular as a new paradigm of classification and learning. SVMs are also recognized as efficient tools for data mining and are popular because of two important factors. First, unlike the other classification techniques, SVMs minimize the expected error rather than minimizing the classification error. Second, SVMs employ the duality theory of mathematical programming to get a dual problem that admits efficient computational methods. SVMs represent a paradigm of classification techniques and include several classes of techniques, which differ among themselves, based on the kernel function and the linear or non-linear separating surfaces between classes.

One cannot overlook the presence of an optimization problem hidden in the classification or clustering tasks. The minimization of error or maximization of the effectiveness of classification are the key ideas in building a decision tree or segmenting the data set into clusters. Instead of minimizing the error, SVMs incorporate structured risk minimization into the classification. By structured risk minimization, we mean minimizing an upper bound on the generalization error. In a supervised classification, we have a training data set and a test data set. The classification error is calculated based on the ratio (or percentage) of misclassified test data items to the whole test data set. However, the classification scheme is generated from the training data set. The training data set and the test data set may be the same or altogether different. It is up to the user to specify these two data sets. On the other hand, the generalization error is less if the test data exhibit a similar classification as the training data. However, the generalization error is high if these two data sets contradict each other in terms of representing class boundaries. SVMs create a classifier with minimized expected probability of error, which means good generalization.

Consider a simple case when two data sets, A and B, are linearly separable. Traditionally, we attempt to discriminate the points in A and B by constructing a separating hyperplane $X^t W = \gamma$, so that the open half space $\{X \mid X \in R^n, X^t W > \gamma\}$ contains mostly the points of A, and the open half space $\{X \mid X \in R^n, X^t W < \gamma\}$ contains mostly the points of B. In other words, we wish to determine W and γ, such that the following two inequalities are satisfied.

$$AW > e\gamma \quad \text{and} \quad BW < e\gamma,$$

where A denotes the matrix corresponding to all X in class A; B denotes a matrix for data in B; and e is the vector of all 1s.

One can see that there can be an infinite number of planes satisfying these conditions. Neural networks attempt to determine the hyperplane satisfying these conditions, such that the error is minimized. On the other hand, SVMs attempt to

determine, among the infinite number of planes, the one that will have smallest generalization error.

We can write the normalized version of the above pair of inequalities as

$$AW \geq e\gamma + e \quad \text{and} \quad BW \leq e\gamma - e.$$

This means that for every X in the class A, we have $X^t W - \gamma \leq +1$ and for every X in class B, we have $X^t W - \gamma < -1$ (Figure 7.13). This represents a pair of parallel hyperplanes separating two classes. The SVM focuses on determining two maximally apart parallel hyperplanes. Thus, SVMs choose the planes that maximize the margin separating two classes. The key idea is that the classifier must reduce the expected classification error and the confidence interval to ensure good generalization. Geometrically, this corresponds to widening the gap between two parallel planes, one supporting class A and the other supporting class B. The wider the gap, the smaller is the generalization error.

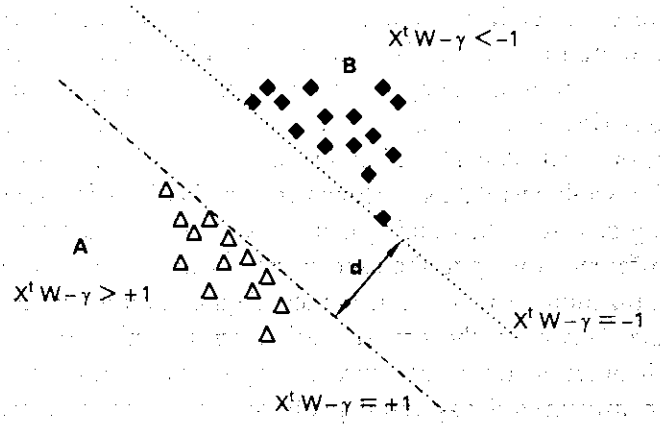


Figure 7.13 Parallel hyperplanes separating two classes. The distance between the two planes is $d = 2/\| (W,\gamma) \|_2$

We say that the hyperplane determined by (W, γ) supports the set A, when every element X of A satisfies $XW \geq \gamma$ (or, $XW \leq \gamma$) and there is at least one element that satisfies this as equality. In order to get two maximally apart parallel hyperplanes, it is necessary that these planes are the supporting surfaces of the respective classes.

In order to determine the hyperplane whose two parallel planes support A and B, so that the gap between these parallel planes is the widest, we formulate the following optimization problem:

$$Maximize_{W,\gamma} \frac{2}{\| (W,\gamma) \|}$$

$$\text{subject to } \begin{cases} AW - e\gamma \geq e \\ -BW + e\gamma \geq e. \end{cases}$$

For the more general case, when the classes are not linearly separable, we formulate the same problem by introducing two additional variables as the following quadratic programming problem:

$$Minimize_{w,\gamma,y,z}(1-\lambda)\|(y,z)\| + \frac{\lambda}{2}\|(W,\gamma)\|$$

s.t.

$$-AW + e\gamma + e \leq y,$$
$$BW - e\gamma + e \leq z,$$
$$y \geq 0, z \geq 0.$$

This is a convex quadratic programming problem. We can consider different norms to construct different linear SVMs. A simple case can be written as follows:

$$Minimize_{w,\gamma,y,z}(1-\lambda)(y^t y + z^t z) + \frac{\lambda}{2}(W^t W + \gamma^2)$$

s.t.

$$-AW + e\gamma + e \leq y,$$
$$BW - e\gamma + e \leq z,$$
$$y \geq 0, z \geq 0.$$

Clearly, for the above quadratic problem, the following condition is satisfied at the optimal point

$$y = (-AW + e\gamma + e)_+,$$
$$z = (BW - e\gamma + e)_+,$$

where $(x)_+$ denotes Max$(x, 0)$.

Thus, the quadratic programming problem can be written as the unconstrained convex quadratic optimization problem, by substituting these values for y and z. The dual of this problem is simple to solve even for high input dimensions and carries very simple constraints. Readers are referred to literature on convex programming problems for deriving the dual of the problem and for the solution techniques. Often, one can even determine a close form solution for the dual. This makes the computation task very simple. It is interesting to study the iterative scheme proposed by Mangasarian [Mangasarian, 2000].

The SVMs differ among themselves on using different norms and different kernel functions while formulating the optimization problem. It is demonstrated that SVMs

are very useful for knowledge discovery, classification and regression analysis. Bennett *et al.* propose a method of using SVMs for decision tree constructions.

Let use consider the data set depicted in Figure 7.14 to demonstrate the use of SVM for the construction of a decision tree. We can employ one SVM to partition the data sets into two subsets: one consisting of $A_1 \cup B_1$, and the other of $A_2 \cup B_2$. This constitutes the first level splitting of the decision tree. For each of the subsets, we can use one more SVM to split and classify the two classes, A and B. Bennett *et al.* show that all the three SVMs can be equivalently formulated as a single optimization problem whose dual is simple to solve. It is also easy to find the primal solution from the dual. Thus, SVM can be utilized to construct a generalized decision tree more efficiently.

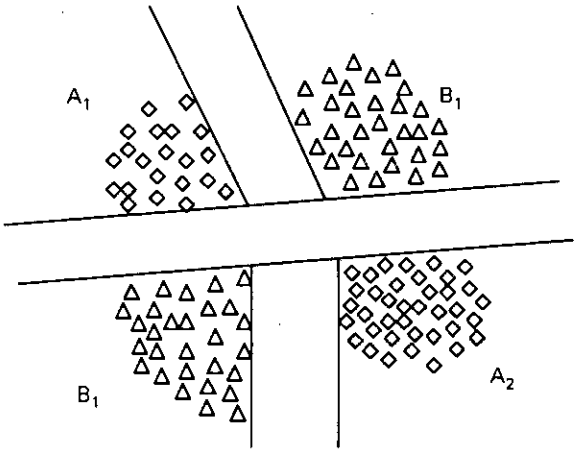


Figure 7.14 Oblique Decision Tree using SVMs.

Joachims demonstrated the applications of SVMs to text categorization, to achieve error rates significantly below those achievable by other known classification techniques. SVMs provide a significant improvement in accuracy, but at the cost of simplicity and time efficiency.

Intuitively, SVMs map non-linearly their n-dimensional input space into a high-dimensional feature space. In this high dimensional feature space, a linear classifier is constructed. Two results make this approach successful:

- The generalization ability of this learning machine depends on the VC dimension of the set of functions that the machine implements, rather than on the dimensionality of the space. A function that describes the data well and belongs to a set with low VC dimension will generalize well, regardless of the dimensionality of the space.
- The construction of the classifier only needs to evaluate an inner product between

two vectors of the training data. An explicit mapping into the high-dimensional feature space is not necessary. In a Hilbert space, inner products have simple kernel representations and, therefore, can be easily evaluated.

7.9 Conclusion

In this chapter, we have discussed four different techniques, namely, Neural Networks, Genetic Algorithms, Rough Set Theory and Support Vector Machines. These subjects are not yet developed to the extent that other techniques like decision trees, clustering or association rules. Nevertheless, these techniques are undoubtedly important and efficient techniques for data mining applications. We envisage that in next few years, these techniques will surely mature and be dealt with as independent topics within the area of data mining. In this chapter, we have not focused on the algorithmic aspects of SVMs, but have only outlined the basic principles.

Further Reading

Neural Networks

There are many excellent books on neural networks outlining various models. There are also many recent research papers which demonstrate the suitability of neural computing for data mining. However, there are not many papers which handle disk-resident training data sets. Kohonen's WEBSOM is one of the pioneering works [Kohonen, 1997]. The case study in the text is taken from [Shalvi].

GA

Goldberg's book is an excellent reference for GA.

Rough Sets

The theory of rough sets has been under continuous development for over 12 years now, and a fast-growing group of researchers and practitioners are interested in this methodology. The theory was proposed by Zdzislaw Pawlak in the 1970s as a result of a long-term program of fundamental research on the logical properties of information systems, carried out by him and a group of logicians from the Polish Academy of Sciences and the University of Warsaw, Poland. A brief introduction on the theory of rough sets is given by the *Electronic Bulletin on Rough Sets* (*EBRS*, 1993).

SVM

The origin of SVMs perhaps goes back to very early research by Vapnik [Vapnik, 1974] and [Vapnik, 1979]. In these two books, the original "Generalized Portrait" Algorithm for constructing separating hyperplanes with an optimal margin is described. In addition, important work in the context of reproducing kernels, related to the SVM methods, has been done by Wahba and co-workers. An introduction to SVM can be found in the articles of Mangasarian and of Vapnik. Benett proposes the use of SVM for an oblique decision tree.

Exercises

1. Describe the principle of neural computing and discuss its suitability to data mining.
2. Discuss the salient features of the genetic algorithm. How can a data mining problem be an optimization problem? How do you use GA for such cases?
3. Discuss the underlying principles of the rough set theory. Discuss the situations when this theory may be useful for data mining.
4. Given a transaction database, define some equivalence relations on the set of attributes or the set of tuples. Find the lower and upper approximation. Is it possible to discover some information from this?
5. How is SVM different from MLP-based classification?
6. Propose a method of implementing MLP with error back-propagation learning, such that the system can learn and classify based on disk-resident data. Discuss the possibility of minimizing the I/O operation during learning.
7. Is it possible to use the perceptron for building decision trees? Please recall that a decision tree also defines a hyperplane for splitting. Distinguish between these methods.

Bibliography

Neural Networks

Fausett L., *Fundamentals of Neural Networks*, Prentice-Hall, 1994.

Gurney K., *An Introduction to Neural Networks*, UCL Press, 1997.

Hassoun M.H., *Fundamentals of Artificial Neural Networks*, MIT Press, Cambridge, MA, 1995.

Hertz J., Krogh A., and Palmer R.G. *Introduction to the Theory of Neural Computation,* Addison-Wesley, Redwood City, California, 1991.

Kohonen Tuevo, "The self-organizing map." In *Proceedings of the IEEE*, **78**, no. 9, pp. 1464–1480, Sept. 1990.

McCulloch W.S., and Pitts W. A Logical Calculus of the Ideas Imminent in Nervous Activity. *Bulletin of Mathematical Biophysics*, 5, 115–133, 1943.

Merkl D., Text data mining. In Dale R., Moisl H. and Somers H (eds.). *A Handbook of NLP: Techniques and applications for the processing of language as text.* Marcel Dekker, 1998.

Mitchell. *Machine Learning*, WCB/McGraw Hill, Boston, 1997.

Shalvi D., and DeClaris N. An Unsupervised Neural Network Approach to Medical Data Mining Techniques, www.eng.umd.edu/medlab/papers/dcsThShort/thpaper1.htm, 1996.

Genetic Algorithms

Goldberg D.E., *Genetic Algorithm in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Company, 1989.

Holland J.H. *Adaptation in Natural and Artificial System* (2nd ed.), Prentice Hall, 1992.

Marmelstein R., and Lamont G. Pattern Classification using a Hybrid Genetic Program-Decision Tree Approach. In *Proceedings of the 3rd Annual Genetic Programming Conference*, 223–231, 1998.

McCallum R., and Spackman K. Using genetic algorithms to learn disjunctive rules from examples. In *Proceedings of the 7th International Conference on Machine Learning*, 149–152, 1990.

Ryan M.D., and Rayward-Smith V.J. The evolution of decision trees. In *Proceedings of the Third Annual Genetic Programming Conference*, 350–358, 1998.

Syswerda G. In *First Workshop on the Foundations of Genetic Algorithms and Classification Systems*, Morgan Kaufmann, 1990.

Rough Sets

Banerjee M., and Chakraborty M.K. Rough Logies: A survey with further directions. In E. Orlowska (ed.) *Rough Sets Analysis*. Physica-Verlag, Heidelberg (1997).

Pawlak Z. Reasoning about Data — A rough set perspective. In Adams, E.W. (ed.) *The Logic of Conditionals, An Application of Probability to Deductive Logic*. D. Reidel Publishing Company, Dordrecht, Boston (1975).

Bandler W., and Kohout L. Fuzzy power sets and fuzzy implication operators. *Fuzzy Sets and Systems*, **4** (1980).

Demri S., and Orlowska E. *Logical Analysis of Indiscernibility*. Institute of Computer Science, Warsaw University of Technology, ICS Research Report 11/96.

Ziarkow W., and Shan N. KDD-R; A comprehensive systems for knowledge discovery using rough sets. In *Proceedings of the International Workshop on Rough Sets and Soft Computing (RSSC '94)*, 1994.

Greco S., Matarazzo B., and Slowinski R. *A New Rough Set Approach to Multi-criteria and Multi-attribute Classification.*

Nguyen S., Hoa S., and Synak P. *Rough Sets in Data Mining: Approximate Description of Decision Classes.*

Wong S.K.M., and Ziarkow W. On learning and evaluation of decision rules in the context of rough sets. In *Proceedings of the International Symposium on Methodologies for Intelligent Systems*, 1986.

SVM

Bennett K.P., and Blue J.A. A support vector machine approach to decision tree. In *IJCNN'98*, 1998.

Bradley P.S., Fayyad U.M., and Mangasarian O.L. Mathematical programming for data mining: Formulations and challenges. *Microsoft Research Tech Report*, MSR-TR-98-04, 1998.

Bradley P.S., and Mangasarian O.L. Massive data discrimination via linear support vector machines, *Tech Report 98-05*, University of Wisconsin, 1998.

Burges C.J.C. A tutorial on support vector machines for pattern recognition, *Data Mining and Knowledge Discovery*, 121–167, 1998.

Cortes C., and Vapnik V.N. Support vector networks. *Machine Learning*, **20**:273–297, 1995.

Joachims T. Text categorization with support vector machines, learning with many relevant features. In *Proc. of ECML-98.*

Kimeldorf G.S., and Wahba G. Some results on Tchebycheffian spline functions. *J. Math. Anal. Applic.*, **33**, 1: 82–95 (1971).

Mangasarian O.L. A generalized support vector machine. *Technical Report*, University of Wisconsin, ftp://ftp.cs.wisc.edu/math-prog/tech-reports/98.14.ps. 1998

Mangasarian O.L., and Musicant D.R. Lagrangian Support Vector Machines. ftp://ftp.cs.wisc.edu/math-prog/tech-reports. 2000

Nashed Z., and Wahba G., Generalized inverses in reproducing kernel spaces: An approach to regularization of linear operator equations. *SIAM J. Math. Anal.* **5**, 6: 974–987, 1974

Vapnik V., and Chervonenkis V. *A Theory of Pattern Recognition* [*in Russian*]. Nauka, Moscow. 1974 (German Translation: W. Wapnik, and A. Tscherwonenkis. *Theorie der Zeichenerkennung*, Akademie-Verlag, Berlin, 1979.

Vapnik V. *Estimation of Dependences Based on Empirical Data [in Russian]*. Nauka, Moscow. 1979. (English translation: 1982, Springer Verlag, New York)

Vapnik V., Golowich S.E., and Smola A. Support vector method for function approximation, regression estimation and signal processing. In *Advances in Neural Information Processing Systems*, **9**, (eds.) Mozer M.C., Jordan M.I. and Petsch T., MIT Press, 1997.

8

WEB MINING

8.1 INTRODUCTION

In recent years we have witnessed an ever-increasing flood of written information, culminating in the advent of massive digital libraries. The world wide web has become a very popular medium of publishing. Though the web is rich with information, gathering and making sense of this data is difficult because publication on the web is largely unorganized. Having learnt the data mining techniques, a question that naturally comes to our minds is whether we can extract implicit, previously unknown information from the massive collection of documents available in the web. And, if so, do we have to design new techniques or we can make use of the standard data mining techniques available? In this chapter, the essential features of web mining are identified. We also discuss the subproblems, where these standard techniques can be employed. It will be shown here that text mining research is the most important aspect of web mining.

The objective of this chapter is to present a brief overview of web mining techniques. Readers may notice that, unlike the earlier chapters, details of the techniques

are not discussed here. We feel that the topic is still in its infancy and hence, it is sufficient at present to study the fundamental principles only and not the techniques. We shall, however, provide sufficient information for a researcher to initiate a research programme in web mining.

8.2 Web Mining

We make use of the web in several ways. As Kosala *et al.* put it, we interact with the web for the following purposes.

Finding Relevant Information

We either browse or use the search service when we want to find specific information on the web. We usually specify a simple keyword query and the response from a web-search engine is a list of pages, ranked based on their similarity to the query. However, today's search tools have the following problems:

- *Low precision*: This is due to the irrelevance of many of the search results. We may get many pages of information which are not really relevant to our query.
- *Low recall*: This is due to the inability to index all the information available on the web. Because some of the relevant pages are not properly indexed, we may not get those pages through any of the search engines.

Discovering New Knowledge from the Web

We can term the above problem as a query-triggered process (retrieval oriented). On the other hand, we can have a data-triggered process that presumes that we already have a collection of web data and we want to extract potentially useful knowledge out of it (data mining-oriented).

Personalized Web Page Synthesis

We may wish to synthesize a web page for different individuals from the available set of web pages. Individuals have their own preferences in the style of the contents and presentations while interacting with the web. The information providers like to create a system which responds to user queries by potentially aggregating information from several sources, in a manner which is dependent on the user.

Learning about Individual Users

It is about knowing what the customers do and want. Inside this problem, there are

subproblems, such as mass customizing the information to the intended consumers or even personalizing it to individual user, problems related to effective web site design and management, problems related to marketing, etc.

Web mining techniques provide a set of techniques that can be used to solve the above problems. Sometimes, web mining techniques provide direct solutions to above problems. On the other hand, web mining techniques can be used as a part of a bigger application that addresses the above problems. However, web mining techniques are not the only tools to handle these problems. Other related techniques from different research areas, such as database (DB), information retrieval (IR), and natural language processing (NLP), can also be used.

Web mining, when looked upon in data mining terms, can be said to have three operations of interests—clustering (e.g., finding natural groupings of users, pages, etc.), associations (e.g., which URLs tend to be requested together), and sequential analysis (e.g., the order in which URLs tend to be accessed). As in most real-world problems, the clusters and associations in web mining do not have clear-cut boundaries and often overlap considerably.

Mining techniques in the web can be categorized into three areas of interest (Figure 8.1), based on which part of the web is to be mined [Madria, 1998]. They are

1. Web content mining,
2. Web structure mining, and
3. Web usage mining.

8.3 Web Content Mining

Web content mining describes the discovery of useful information from the web contents. However, what comprise the web contents could encompass a very broad range of data.

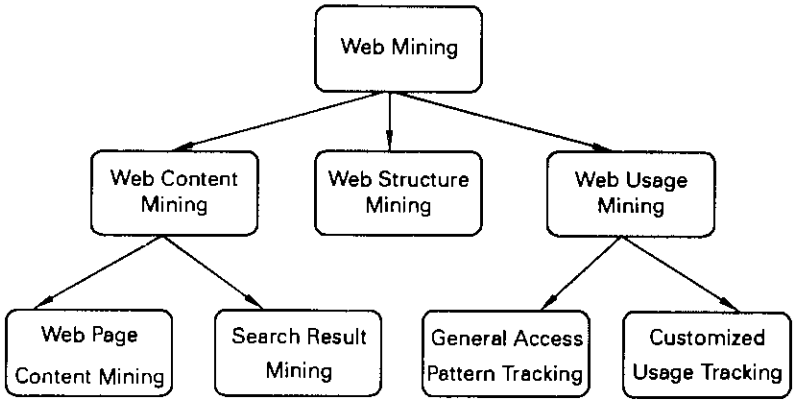


Figure 8.1 Web Mining Tasks

The web contains many kinds of data. We see that much of the government information are gradually being placed on the web in recent years. We also know the existence of Digital Libraries that are also accessible from the web. Many commercial institutions are transforming their businesses and services electronically. We cannot ignore another type of web content—the existence of web applications, so that the users could access the applications through web interfaces. Many applications and systems are being migrated to the web and many types of applications are emerging in the web environment itself. Some of the web content data are hidden data, and some are generated dynamically as a result of queries and reside in the DBMSs. These data are generally not indexed.

Basically, the web content consists of several types of data such as textual, image, audio, video, metadata, as well as hyperlinks. Recent research on mining multi-types of data is termed as *multimedia data mining*. This line of research is yet to receive proper attention and most of the efforts on web content mining are concentrated on the text or hypertext contents. The textual parts of web content data consist of unstructured data such as free texts, semi-structured data such as HTML documents, and more structured data such as data in the tables or database-generated HTML pages. Undoubtedly, much of the web content data is unstructured, free text data. As a result, the techniques of text mining can be directly employed for web content mining in such cases. We shall study text mining techniques in the later part of this chapter.

8.4 Web Structure Mining

Web structure mining is concerned with discovering the model underlying the link structures of the web. It is used to study the topology of the hyperlinks with or without the description of the links. This model can be used to categorize web pages and is useful to generate information such as the similarity and relationship between different web sites. Web structure mining can be used to discover authority sites for the subjects and overview (or hub) sites for the subjects that point to many authorities. It may be noted that while web content mining attempts to explore the structure within a document (intra-document structure), web structure mining studies the structures of documents within the web itself (inter-document structure).

We can view any collection, V, of hyperlinked pages as a directed graph $G = (V, E)$: the nodes correspond to the pages, and a directed edge $(p, q) \in E$ indicates the presence of a link from p to q. We say that the out-degree of a node p is the number of nodes to which it has links, and the in-degree of p is the number of nodes that have links to it. If $W \subseteq V$ is a subset of the pages, we use $G[W]$ to denote the graph induced on W—its nodes are the pages in W, and its edges correspond to all the links between the pages in W.

Some algorithms have been proposed to model web topology such as HITS,

PageRank, CLEVER; and these models are mainly applied as a method to calculate the quality rank or relevancy of each web page. We shall outline below some of the techniques that are useful in modelling web topology.

PageRank

The motivation is from academic citation literature, where the importance of a document is measured by counting citations or backlinks to a given document. This gives some approximation of a document's importance or quality. This concept is extended to web pages. The intuitive justification is that a page can have a high PageRank if there are many pages that point to it, or if there are some pages that point to it which have a high PageRank. Intuitively, pages that are well cited from many places around the web are worth looking at. Also, pages that have perhaps only one citation from something like the Yahoo! homepage are also generally worth looking at. PageRank handles both these cases and everything in between, by recursively propagating weights through the link structure of the web.

PageRank is defined as follows:
We assume page A has pages T_1, ..., T_n, which point to it (i.e., are citations). The parameter d is a damping factor which can be set between 0 and 1 and is usually set to 0.85. out_deg(A) denotes the number of links going out of page A (out-degree of A).

Definition 8.1 PageRank

The PageRank of a page A is given as follows:

$$PR(A) = (1-d) + d\left(\sum_{i=1}^{n} \frac{PR(T_i)}{out\_\deg(T_i)}\right).$$

PageRank or $PR(A)$ can be calculated using a simple iterative algorithm, and corresponds to the principal eigenvector of the normalized link matrix of the web. Let n be the number of documents we have. We define the link matrix M, where the M_{ij} entry is $1/n_j$ if there is a link from document j to document i, otherwise M_{ij} is 0. n_j is the number of the forward link of document j (out degree of j). Then we can compute the PageRank on the graph which is the dominant eigenvector of the matrix A.

Note that the PageRanks form a sort of a probability distribution over the web pages. PageRank also provides models of user behaviour. Consider a surfer starting from a web page at random and who keeps clicking on links, never hitting "back", but eventually be gets bored and switches to another random page. The probability that the surfer visits a page is its PageRank. The damping factor d is to model the probability that at each page the surfer would get bored and request another random page.

Social Network

Social network analysis is yet another way of studying the web link structure. It uses an exponentially varying damping factor. Web structure mining utilizes the hyperlinks structure of the web to apply social network analysis, to model the underlying links structure of the web itself. The social network studies ways to measure the relative *standing* or importance of individuals in a network. The same process can be mapped to study the link structures of the web pages. The basic premise here is that if a web page points a link to another web page, then the former is, in some sense, endorsing the importance of the latter. Links, in this network, may have different weights, corresponding to the strength of endorsement.

Kautz *et al.* in a pioneering work on web structure mining, *The Hidden Web*, propose a measure of *standing* of a node based on path counting. They carry out social network analysis to model the network of AI researchers. The standing of a node(page) can be defined as follows.

Definition 8.2 Standing of a Node

For nodes p and q, let $P_{pq}^{(r)}$ denote the number of paths of length exactly r from p to q. Let $b < 1$ be a constant, chosen to be small enough so that $Q_{pq} = \Sigma_r b^r P_{pq}^{(r)}$ converges. One can view b^r as the damping factor. Please note that the damping factor varies with the length of the path.

The *standing* of the node q, σ_q is defined as $\Sigma_p Q_{pq}$.

Transverse and Intrinsic Links

Kleinberg discusses a heuristic method of giving weightage to links. A link is said to be a *transverse link* if it is between pages with different domain names, and an *intrinsic link* if it is between pages with the same domain name. Here, by "domain name", we mean the first level in the URL string associated with a page. Since intrinsic links very often exist purely to navigate the infrastructure of a site, they convey much less information than transverse links about the importance of the pages to which they point. Thus, while computing the PageRank or standing of a page, the intrinsic links need not be taken into account. Kleinberg proposes to delete all intrinsic links from the graph, keeping only the edges corresponding to transverse links. This is a very simple but effective heuristic

Reference Nodes and Index Nodes

Botafogo *et al.* propose another way of ranking pages. They define the notion of index nodes and reference nodes.

Definition 8.3 Index Node

An index node is a node whose out-degree is significantly larger than the average out-degree of the graph.

Definition 8.4 Reference Node

A reference node is a node whose in-degree is significantly larger than the average in-degree of the graph.

Clustering and Determining Similar pages

For determining the collection of similar pages, we need to define the similarity measure between pages. There can be two basic similarity functions.

Definition 8.5 Bibliographic Coupling

For a pair of nodes, p and q, the bibliographic coupling is equal to the number of nodes that have links from both p and q.

Definition 8.6 Co-citation

For a pair of nodes, p and q, the co-citation is the number of nodes that point to both p and q.

We give a simple graph of clustering techniques for clustering the web documents based on the web structure.

We first identify the *influential* pages that are referred by substantially large number of pages. The next step is to create a soft cluster around each of the influential pages based on citation count. The pages are assigned to the soft cluster if they are co-cited along with the influential page. After creating those soft clusters, the similarity between the soft clusters are calculated and based on the values of the similarity measure, certain soft clusters are merged in the hierarchical agglomorative clustering principle.

In order to determine the set of influential pages, it is necessary to define a threshold λ and identify the nodes whose degree exceeds the value λ. There are different variations to this approach. One can only count the out-degree or in-degree; can count either the transverse or intrinsic degrees. In order to build a soft cluster around an influential node v, we identify all other nodes x, such that there is at least one node y that cites both x and v. In some cases we can take into account both *bibliographic* and *co-citation couplings*. The similarity measure between two sub-clusters S_x and S_y is computed as

$$\frac{|S_x \cap S_y|}{|S_x \cup S_y|}.$$

EXAMPLE 8.1

Let us understand the working of the algorithm based on the set of URLs taken from www.cora.jprc.com (see Addendum at the end of this chapter). In the subsequent discussion we shall refer to these URLs by their respective serial numbers. There are 101 URLs. We assume λ to be 8. Based on the adjacency list of the graph, Figure 8.2, the set of influential nodes are identified as nodes 1, 2, 3 and 5. The soft clusters are computed by co-citation links, as shown in Figure 8.3. Figure 8.4 shows the merging of clusters based on similarity function.

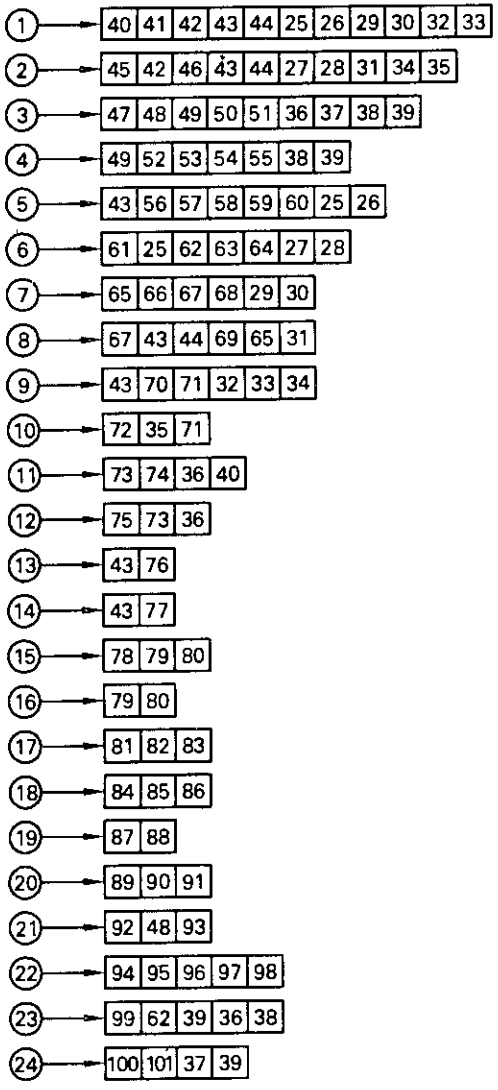


Figure 8.2 Adjacency List of the Graph Formed by the URLs

Soft Cluster 1: {7 65 66 67 68 29 30 8 43 44 69 31 9 70 71 32 33 34 1 40 41 42 25 26}

Soft Cluster 2: {6 61 25 62 63 64 27 28 8 67 43 44 69 65 31 9 70 71 32 33 **34** **2** 45 42 46 35}

Soft Cluster 3: {4 49 52 53 54 55 38 39 23 99 62 36 24 100 101 37 **3** 47 48 50 51}

Soft Cluster 4: {**5** 43 56 57 58 59 60 25 26}

Figure 8.3 The Set of Soft Cluster. Number in Boldface Indicates the Influential Nodes.

{5 43 56 57 58 59 60 25 26 6 61 62 63 64 27 28 8 67 44 69 65 31 9 70 71 32 33
34 2 45 42 46 35 7 66 68 29 30 1 40 41}

{4 49 52 53 54 55 38 39 23 99 62 36 24 100 101 37 3 47 48 50 51}

Figure 8.4 Merged Clusters Based on Similarity

8.5 Web Usage Mining

Web usage mining deals with studying the data generated by the web surfer's sessions or behaviours. Note that the web content and structure mining utilize the real or primary data on the web. On the contrary, web usage mining mines the secondary data derived from the interactions of the users with the web. The secondary data includes the data from the web server access logs, proxy server logs, browser logs, user profiles, registration data, user sessions or transactions, cookies, user queries, bookmark data, mouse clicks and scrolls, and any other data which are the results of these interactions.

This data can be accumulated by the web server. Analyses of the web access logs of different web sites can facilitate an understanding of the user behaviour and the web structure, thereby improving the design of this colossal collection of information. There are two main approaches in web usage mining driven by the applications of the discoveries.

General Access Pattern Tracking

This is to learn user navigation patterns (impersonalized). The general access pattern tracking analyzes the web logs to understand access patterns and trends. These analyses can shed better light on the structure and grouping of resource providers.

Customized Usage Tracking

This is to learn a user profile or user modelling in adaptive interfaces (personalized). Customized usage tracking analyzes individual trends. Its purpose is to customize web sites to users. The information displayed, the depth of the site structure, and the format of the resources can all be dynamically customized for each user over time, based on their access patterns.

It is important to note that the success of such applications depends on what and how much valid and reliable knowledge one can discover from the large, raw log data.

Current web servers store limited information about the accesses. Some scripts which are custom-tailored for some sites, may store additional information. However, for effective web usage mining, an important cleaning and data transformation step may be needed before analysis.

The mining techniques for web usage mining can be classified into two commonly used approaches. The first approach maps the usage data of the web server into relational tables before a (traditional) data mining technique is performed. In general, typical data mining methods (such as clustering and classification) could be used to mine the usage data after the data have been pre-processed to the desired form. However, modifications of the typical data mining methods are also used, such as composite association rules, an extension of a traditional sequence discovery algorithm MIDAS, and hypertext probabilistic grammars. The second approach uses the log data directly by utilizing special pre-processing techniques. The web usage data can also be represented with graphs. Chakrabarty *et al.* propose a method of extracting themes from the bookmark files of a community of surfers. Often, the web usage mining uses some background or domain knowledge, such as navigation templates, web content, site topology, concept hierarchies, and syntactic constraints.

8.6 Text Mining

Due to the continuous growth of the volumes of text data, automated extraction of implicit, previously unknown, and potentially useful information becomes more necessary to properly utilize this vast source of knowledge. Text mining, therefore, corresponds to the extension of the data mining approach to textual data and is concerned with various tasks, such as extraction of information implicitly contained in collection of documents, or similarity-based structuring.

Text collection, in general, lacks the imposed structure of a traditional database. The text expresses a vast range of information, but encodes the information in a form that is difficult to decipher automatically. The data mining techniques that have been discussed in the earlier chapters are essentially designed to operate on structured databases. When the data is structured it is easy to define the set of items and hence, it becomes easy to employ the traditional mining techniques. Identifying individual items or terms is not so obvious in a textual database. Thus, unstructured data, particularly free-running text, places a new demand on data mining methodology. Specific techniques, called text mining techniques, have to be developed to process the unstructured textual data to aid in knowledge discovery.

The inherent nature of textual data, namely unstructured characteristics, motivates the development of separate text mining techniques. One way is to impose a structure on the textual database and use any of the known data mining techniques meant for structured databases. The other approach would be to develop a very specific

technique for mining that exploits the inherent characteristics of textual databases. Irrespective of the approach chosen for text mining, one cannot ignore the close interactions of other related subjects, such as computational linguistics, natural language processing, and information retrieval. In this chapter, we shall briefly discuss the relationships among these fields of research. A detailed discussion on these topics is beyond the scope of this book.

Other Related Areas

It is important to study the relationship between areas like information retrieval (IR), information extraction (IE), and computational linguistics with text data mining. We are just beginning to see the emergence of a text-based knowledge discovery system, while the information retrieval community has been tackling the querying issues for over 30 years.

Information Retrieval

IR is concerned with finding and ranking documents that match the users' information needs. The way of dealing with textual information by the IR community is a keyword-based document representation. A body of text is analyzed by its constituent words, and various techniques are used to build the core words for a document. The goals are

- To find documents that are similar, based on some specification of the user.
- To find the right index terms in a collection, so that querying will return the appropriate document.

What is still missing are tools providing assistance for an explorative search and pattern finding. It may be noted that the IR problem is not so much that the desired information coexists with many other valid pieces of information; the problem is one of homing in on what is currently of interest to the user. It is rarely the process of pattern finding or of exploration.

Actually, IR is the automatic retrieval of all relevant documents while at the same time retrieving as few of the non-relevant ones as possible. IR has the primary goals of indexing the text and searching for useful documents in a collection. However, recent trends in IR research include modelling, document classification and categorization, user interfaces, data visualization, filtering, etc.. The task that can be considered to be an instance of text mining is document classification or categorization, which could be used for indexing. Viewed in this respect, text mining is part of the IR process. However, we should note that not all of the indexing tasks use data mining techniques.

Information Extraction

IE has the goal of transforming a collection of documents, usually with the help of an

IR system, into information that is more readily digested and analyzed. IE extracts relevant facts from the documents, while IR selects relevant documents. Thus, in general, IE works at a finer granularity level than IR does on the documents. Most IE systems use machine learning or data mining techniques to learn the extraction patterns or rules for documents semi-automatically or automatically. Within this view, text mining is part of the IE process.

The results of the IE process could be in the form of a structured database, or could be a compression or summary of the original text or documents. One could view for the former that IE is a kind of pre-processing stage in the text mining process, which is the step after the IR process and before data mining techniques are performed. In a similar view, IE can also be used to improve the indexing process, which is part of the IR process. In an another viewpoint, IE is an instance of text mining, since the summary or the compressed form of a document is a form of information that did not exist before.

Computational Linguistics

Corpus-based computational linguistics computes statistics over large text collections in order to discover useful patterns. These patterns are used to inform algorithms for various subproblems within natural language processing, such as part-of-speech tagging, word-sense disambiguation, etc. The aims of text data mining are also rather similar to this. However, within the computational linguistics framework, patterns are discovered to aid other problems within the same domain, whereas text data mining is aimed at discovering unknown information for different applications.

8.7 Unstructured Text

Unstructured documents are free texts, such as news stories. Traditionally, most of research uses bags of words to represent unstructured documents and extract different features from it.

Features

For an unstructured document, features are extracted to convert it to a structured form. Some of the important features are listed below.

Word Occurrences

The bag of words or vector representation takes single words found in the training corpus as features ignoring the sequence in which the words occur. This representation is based on the statistic about single words in isolation. Such a feature is said to be Boolean, if we consider whether a word either occurs or does not occur in a document.

The feature is said to be frequency based if the frequency of the word in a document is taken into consideration.

Stop-words

The feature selection includes removing the case, punctuation, infrequent words, and stop words. A good site for the set of stop-words for the English language is www.dcs.gla.ac.uk/idom/ir_resources/linguistic_util/stop_words

Latent Semantic Indexing

Latent Semantic Indexing (LSI) transforms the original document vectors to a lower dimensional space by analyzing the correlational structure of terms in the document collection, such that similar documents that do not share terms are placed in the same topic.

Stemming

Stemming is a process which reduces words to their morphological roots. For example, the words "informing", "information", "informer", and "informed" would be stemmed to their common root "inform", and only the latter word is used as the feature instead of the former four.

n-gram

Other feature representations are also possible, such as using information about word positions in the document, or using n-grams representation (word sequences of length up to n) (In WEBSOM).

Part-of-speech (POS)

One important feature is the POS. There can be 25 possible values for POS tags. Most common tags are noun, verb, adjective and adverb. Thus, we can assign a number 1, 2, 3, 4 or 5, depending on whether the word is a noun, verb, adjective, adverb or any other, respectively.

Positional Collocations

The values of this type of feature are the words that occur one or two position to the right or left of the given word.

Higher Order Features

Other features include phrases, document concept categories, terms, hypernyms, named entities, dates, email addresses, locations, organizations, or URLs. These features could be reduced further by applying some other feature selection techniques, such as information gain, mutual information, cross entropy, or odds ratio.

Once the features are extracted, the text is represented as structured data, and traditional data mining techniques can be used. The techniques include discovering frequent sets, frequent sequences and episode rules. We describe below the preprocessing stage to fund frequent episodes.

8.8 Episode Rule Discovery for Texts

Ahonen *et al.* propose to apply sequence mining techniques for text data. (We shall discuss in detail the techniques of sequence mining later on.) They consider text as sequential data which consists of a sequence of pairs (feature vector, index), where the feature vector is an ordered set of features and the index contains information about the position of the word in the sequence. A feature can be any of the textual features described above.

Define a text episode as a pair $\alpha = (V, \leq)$, where V is a collection of feature vectors and $\leq$ is a partial order on V. Given a text sequence S, a text episode $\alpha = (V, \leq)$ occurs within S if there is a way of satisfying the feature vectors in V, using the feature vectors in S so that the partial order $\leq$ is respected. In other words, the feature vectors of V can be found within S in an order that satisfies $\leq$.

For example, the text *Pathfinder photographs Mars* can be represented as

((*pathfinder*_noun_singular, 1), (*photographs*_verb_singular, 2), (*Mars*_noun_singular, 3))

Similarly, the text *knowledge discovery in databases* can be represented as the sequence

((*knowledge*_noun_singular, 1), (*discovery*_noun_singular, 2), (*in*_preposition, 3), (*databases*_noun_plural, 4))

Instead of considering all occurrences of the episode, a restriction is set that the episode must occur within a prespecified window of size, w. Thus, we examine the substrings S' of S such that the difference of the indices in S' is at most w.

For $w = 2$, the subsequence (*knowledge*_noun_singular, *discovery*_noun_singular) is an episode contained in the window, but the subsequence (*knowledge*_noun_singular, *databases*_noun_plural) is not contained within the window.

The support of α in S is defined as the number of minimal occurrences of α in S. With this formalism, the episode discovery technique of sequence mining can be used to discover frequent episodes in a text. Ahonen *et al.* [Ahonen, 1998, 1999] also propose a technique of discovering maximal frequent subsequences in the text.

8.9 Hierarchy of Categories

When a user enters a query into a search engine, the system often brings back many different pages. It is then necessary to organize the documents into meaningful groups. There are many different ways in which we can show how a set of documents are related to one another. One way is to group together all documents written by the same author, or all documents written in the same year, or published by the same publisher. We can group them according to subject matter as well. Libraries organize some of

their information this way, using classification systems like the Dewey Decimal.

A problem with assigning documents to single categories within a hierarchy (as seen in, for example, Yahoo), is that most documents discuss several different topics simultaneously. A better solution is to describe documents by a set of categories as well as attributes (such as source, date, genre, and author), and provide good interfaces for manipulating these labels.

For this purpose, Feldman *et al.* propose an elegant data structure of concept hierarchy. Concept hierarchy is a directed acyclic graph of concepts, where each of the concept is identified by a unique name. An arc from concept a to b denotes that a is a more general concept than b. We can tag the text with concepts. Each text document is tagged by a set of concepts that correspond to its content.

Tagging a document with a concept implicitly entails its tagging with all the ancestors of the concept hierarchy. It is, therefore, desired that a document should be tagged with the lowest concepts possible. The method to automatically tag the document to the hierarchy is a top-down approach. An evaluation function determines whether a document currently tagged to a node can also be tagged to any of its child nodes. If so, then the tag moves down the hierarchy till it cannot be moved any further.

The outcome of this process is a hierarchy of documents and, at each node, there is a set of documents having a common concept associated with the node. The hierarchy of documents resulting from the tagging process is useful for many text mining process. It is assumed that the hierarchy of concepts is known a *priori*. We can even have such a hierarchy of documents without a concept hierarchy, by using any hierarchical clustering algorithm which results in such a hierarchy.

Popescul *et al.* pose a related problem of tagging key words to the set of documents arranged in a hierarchy. The method is a two-phase principle. It starts with a bag of key words at the leaf level and moves up the hierarchy. The set of key words for a non-leaf node is obtained by combining all the key words to all its child nodes. After finding the set of key words for the root node, the process starts with a top-down approach. If a key word at any node is also equally probable for all of its child nodes, then the key word is associated with the current (parent) node and not with any of the child nodes. Otherwise, if the key word is more probable for a child node, it is moved down to the most probable set of child nodes.

8.10 Text Clustering

Text clustering is another important task of text mining. Once the features of an unstructured text are identified or the structured data of the text is available, text clustering can be done by employing any of the clustering techniques discussed in Chapter 5. One popular text clustering algorithm is Ward's Minimum Variance method. It is an agglomerative hierarchical clustering technique and it tends to

generate very compact clusters. We shall discuss the essential features of this method.

We can take either the Euclidean metric or Hamming distance as the measure of dissimilarities between feature vectors. The clustering method begins with n clusters, one for each text. At any stage, two clusters are merged to generate a new cluster. The clusters, C_k and C_l, are merged to get a new cluster C_{kl} based on the following criterion:

$$V_{kl} = \text{MIN}_{i,j} V_{ij}$$

$$V_{ij} = \frac{\| \overline{x_i} - \overline{x_j} \|^2}{\frac{1}{n_i} + \frac{1}{n_j}},$$

where x_i is the mean value of the dissimilarity for the cluster C_i, and n_i is the number of elements in this cluster.

Scatter/Gather

It is a method of grouping the documents using clustering. The scatter/gather interface uses text clustering as a way to group documents, according to the overall similarities in their content. Scatter/gather is so named because it allows the user to scatter documents into clusters, or groups, then gather a subset of these groups and re-scatter them to form new groups. Each cluster in scatter/gather is represented by a list of topical terms, that is, a list of words that attempt to give the user the gist of what the documents in the cluster are about. The user can also look at the titles of the documents in each group. The documents in the cluster can have other representations as well, such as summaries. If a cluster still has too many documents, the user can re-cluster the documents in the cluster; that is, re-group that subset of documents into still smaller groups. This re-grouping process tends to change the kinds of themes of the clusters, because the documents in a subcollection discuss a different set of topics than all the documents in the larger collection.

8.11 Conclusion

In this chapter, we have outlined three different modes of web mining, namely web content mining, web structure mining and web usage mining. Needless to say, these three approaches cannot be independent, and any efficient mining of the web would require a judicious combination of information from all the three sources. Clustering of hyperlinked documents can rely on a combination of textual and link-based information. Similarly, information about the web structure would greatly enhance the capability of web usage mining. We have noted that web content mining mostly

concentrates on text mining, and textual content in the web can be in an unstructured, semi-structured or structured form. For an unstructured text, features are extracted to view the document in a structured form. Similarly, for unstructured text such as XML, features can be extracted for mining purposes. Recently, Garofalakis *et al.* propose a sequence mining based method to extract document-type description for a set of XML pages. We give below an extensive bibliography for the readers. Merkl proposes a neural network-based text for data mining. Kohonen and his group's pioneering work resulted in WEBSOM; a self-organizing map model for document mining.

Further Reading

Soumen Chakrabarti's [Chakrabarti, 2000] work on mining hypertext makes excellent reading. PageRanking and web search information can be got from Brin [Brin, 1998]. Readers are referred to Botafogo [Botafogo, 1993] for text clustering techniques. The first proposal of text mining is due to Feldman [Feldman, 1995]. Marti Hearst's [Hearst, 1999] work on text mining is also important. This gives a clear understanding of how text mining is not just any other data mining. Kosala's survey on web mining is yet another interesting source of information [Kosala, 2000]. Brin and Page[Brin, 1998] give a good discussion on PageRank. A landmark paper by Kautz, "The Hidden Web" introduces the application of social networks. Cooley's paper [Cooley, 2000] is yet another source of information about web usage mining. Example 8.1 is the part of the work of my masters student M. Prabhavathamma.

Exercises

1. What are the different types of web mining?
2. How is web usage mining different from web structure mining and web content mining?
3. What is the underlying principle of *The Hidden Web*?
4. How is text mining related to web mining? What are the techniques of text mining?
5. Discuss the relationship between text mining and information retrieval and information extraction.
6. What is PageRank? How is it computed?
7. What is concept hierarchy? How is it related to web mining?
8. Discuss the principles underlying text clustering.
9. How do you extract structures from unstructured text data? What features are extracted in this process?
10. Based on the details given in Chapter 5, discuss the suitability of some of the clustering algorithms for text clustering.

11. Which frequent itemset mining is suitable for text mining?
12. What are the differences between mining techniques of structured data, semi-structured data and unstructured data?

BIBLIOGRAPHY

Ahonen H., Heinonen O., Klemettinen M., and Verkamo A. Applying data mining techniques for descriptive phrase extraction in digital document collections. In *Advances in Digital Libraries (ADL'98)*, Santa Barbara, California, USA, April 1998.

Ahonen H., Heinonen O., Klemettinen M., and Verkamo A. Applying data mining techniques in text analysis. http://www.cs.helsinki,fi/~hahonen

Baeza-Yates R., and Berthier Ribeiro-Neto E. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Company, 1999.

Borges J., and Levene M. Mining association rules in hypertext databases. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD-98)*, New York City, New York, USA, August 27–31, 1998.

Borges J., and Levene M. Data mining of user navigation patterns. In *Proceedings of the WEBKDD'99 Workshop on Web Usage Analysis and User Profiling*, San Diego, CA, USA, August 15, 1999, pp. 31–36.

Botafogo R.A. Cluster analysis for hypertext systems. *ACM-SIGIR*, 1993.

Brin S., and Page L. The anatomy of a large-scale hypertextual web search engine. In *7th International World Wide Web Conference*, Brisbane, Australia, 1998.

Buchner A., Baumgarten M., Anand S., Mulvenna M., and Hughes J. Navigation pattern discovery from internet data. In *Proceedings of the WEBKDD'99 Workshop on Web Usage Analysis and User Profiling*, San Diego, CA, USA, August 15, 1999.

Chakrabarti S., Dom B., and Indyk P. Enhanced hypertext categorization using hyperlinks. In L.M. Haas and A. Tiwary (eds.), SIGMOD 1998, In *Proceedings ACM SIGMOD International Conference on Management of Data*, Seattle, Washington, USA, ACM Press, June 2–4, 1998, pp. 307–318.

Chakrabarti S., Dom B., Gibson D., Kleinberg J., Kumar S., Raghavan P., Rajagopalan S., and Tomkins A. Mining the link structure of the world wide web. *IEEE Computer*, **32**(8):60–67, 1999.

Chakrabarti S. Data mining for hypertext: A tutorial survey. *ACM SIGKDD Explorations*, 1(2):111, 2000.

Chakrabarti S., and Batterywala Y. Mining themes from bookmarks. In *ACM SIGKDD 200 Workshop on Text mining*, 2000.

Cooley R., Mobasher B., and Srivastava J. Web mining: Information and pattern discovery on the world wide web. In *Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'97), 1997.*

Cooley R., Mobasher B., and Srivastava J. Data preparation for mining world wide web browsing patterns. *Knowledge and Information Systems*, **1**(1), 1999.

Cooley R.W. Web usage mining: Discovery and application of interesting patterns from web data. PhD Thesis, Dept. of Computer Science, University of Minnesota, May 2000.

Cutting D.R., Karger D., Pederson J, and Tukey J.W. Scatter/Gather: A cluster-based approach to browsing large document collections. In *Proceedings of 15th Annual International ACM/SIGI Conference*, Copenhagen, Denmark, June 1992, pp. 318–329.

Deerwester S., Dumais S., Furnas G., Landauer T., and Harshman R. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, **41**(6), 1990, pp. 391–407.

Feldman R. and Dagan I. Knowledge discovery in textual databases (KDT). In *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining (KDD-95)*, Montreal, Canada, 1995, pp. 112–117.

Feldman R., Fresko M., Kinar Y., Lindell Y., Liphstat O., Rajman M., Schler Y., and Zamir O. Text mining at the term level. In *Principles of Data Mining and Knowledge Discovery, 2nd European Symposium, PKDD '98*, **1510** of Lecture Notes in Computer Science, Springer, 1998, pp. 56–64.

Garofalakis M.N., Rastogi R., Seshadri S., and Shim K. Data mining and the web: Past, present and future. In *Workshop on Web Information and Data Management*, 1999, pp. 43–47.

Garofalakis M.N., Rastogi R., Seshadri S., and Shim K. XTRACT: A system for extracting document type descriptors from XML documents. *SIGMOD 2000*, 2000.

Hearst M.A. Untangling text data mining. In *Proceedings of ACL'99: the 37th Annual Meeting of the Association for Computational Linguistics*, 1999.

Honkela T., Kaski S., Lagus K., and Kohonen T. WEBSOM—self-organizing maps of document collections. In *Proceedings of the Workshop on Self-Organizing Maps 1997 (WSOM'97)*, 1997, pp. 310–315.

Kautz H., Selman B., and Shah M. The hidden web. *AI magazine*, **18**(2), 1997, pp.27–36.

Kleinberg J.M. Authoritative sources in a hyperlinked environment. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, 1998, pp. 668–677.

Koasala R., and Blockeel H. Web mining research: A survey. *ACM SIGKDD*, **2**(1), 2000, pp. 1–15.

Lent B., Agrawal R., and Srikant R. Discovering trends in text databases. In *Proceedings of the 3rd International Conference On Knowledge Discovery and Data Mining (KDD 1997)*, 1997, pp. 227–230.

Madria S.K., Bhowmick S.S., Ng W.K., and Lim E.-P. Research issues in web data mining. In *Proceedings of Data Warehousing and Knowledge Discovery, 1st International Conference, DaWaK '99*, 1999, pp. 303–312.

Mladenic D. and Grobelnik M. Feature selection for unbalanced class distribution and naive Bayes distribution. In *Proceedings of the 16th International Conference on Machine Learning ICML-99*, 1999, pp. 258–267.

Masand B. and Spiliopoulou M. Webkdd-99: Workshop on web usage analysis and user profiling. *SIGKDD Explorations*, **1**(2), 2000.

Merkl D. Text data mining. In Dale R., Motel H., and Somers H. (eds.), A *Handbook of Natural Language Processing: Techniques and Applications for the Processing of Language as Text.* New York, Marcel Dehkker, 1998.

Paliouras G., Papatheodorou C., Karkaletsis V., Tzitziras P., and Spyropoulos C. D. Large-scale mining of usage data on web sites. In *AAAI 2000 Spring Symposium on Adaptive User Interfaces*, 2000.

Peleato R.A., Chappelier J-.C., and Rajman M. Using information extraction to classify newspaper advertisements. *IADT2000*, 2000.

Rajman M., and Besancon R. Text mining—knowledge extraction from unstructured textual data. In *6th Conference of International Federation of Classification Societies, IFCS'98*, pp. 473–480, 1998.

Spiliopoulou M. Data mining for the web. In *Principles of Data Mining and Knowledge Discovery, 2nd European Symposium, PKDD '99*, pp. 588–589, 1999.

Srivastava J., Cooley R., Deshpande M., and Tan P.-N. Web usage mining: Discovery and applications of usage patterns from web data. *SIGKDD Explorations*, 1–2, 2000.

Tan A.-H. Text mining: The state of the art and the challenges. In *Proceedings of the Pacific Asia Conference on Knowledge Discovery and Data Mining PAKDD'99*, Workshop on Knowledge Discovery from Advanced Databases, 1999, pp. 65–70.

Vaithyanathan S. Introduction: Data mining on the internet. *Artificial Intelligence Review*, **13** (5/6), 1999, pp.343–344.

Yang Y. An evaluation of statistical approach to text categorization. *Information Retrieval Journal*, 1999.

Zaiane O.R., Han J., Li Z.-N., Chee S.H., and Chiang J. Multimediaminer: A system prototype for multimedia data mining. In *Proceedings ACM SIGMOD International Conference on Management of Data*, 1998, pp. 581–583.

ADDENDUM

1. www.cs.washington.edu/research/jair/home.html
2. www.cs.cmu.edu/afs/cs.cmu.edu/project/ai-epository/ai/html/air.html
3. www.dcs.gla.ac.uk/Keith/Preface.html
4. ai.bpa.arizona.edu/papers/mlir93/mlir93.html
5. mevard.www.media.mit.edu/groups/agents/
6. www.cs.bham.ac.uk/~amw/agents/
7. www.research.microsoft.com/research/datamine/
8. www.cs.bham.ac.uk/~anp/TheDataMine.html
9. www.ghgcorp.com/clips/CLIPS.html
10. www.nal.usda.gov/wqic/Bibliographies/eb9608.html
11. www.chesslab.com/PositionSearch.html
12. www.pricingcentral.com/fun/videogames.htm
13. www.ics.uci.edu/~mlearn/Machine-Learning.html
14. www.cs.cmu.edu/~tom/mlbook.html
15. www.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/html/faqs/ai/nlp/top.html
16. www.ai.univie.ac.at/oefai/nlu/
17. www.ifla.org/II/metadata.htm
18. www.math.utah.edu/~beebe/digital-libraries.html
19. www.nla.gov.au/nla/staffpaper/cathro4.html
20. www.public-domain.org/database/database.html
21. www.mri.mq.edu.au/~einat/web_ir/
22. www.sciam.com/0397issue/0397resnick.html
23. www.glue.umd.edu/enee/medlab/filter/filter.html
24. www.w3.org/DesignIssues/Filtering.html
25. ai.iit.nrc.ca/subjects/Agents.html
26. www.spectrum.ieee.org/webwatch/web07-31-00.html
27. www-cia.mty.itesm.mx/~lgarrido/Repositories/IA/agents.html
28. www.isis.vt.edu/~fanjun/text/ail.html
29. www-ai.cs.uni-dortmund.de/EVENTS/ICAI99/
30. www.mli.gmu.edu/michalski/teaching/inft910S99.html
31. www.kdnuggets.com/news/97/n04.html
32. www.cslab.ece.ntua.gr/~kblekas/expert.html
33. www.yahoo.com/Science/Computer_Science/Artificial_Intelligence/Expert_Systems/
34. www.gslis.utexas.edu/~palmquis/courses/ai96.html
35. www.rware.demon.co.uk/ail.html
36. citeseer.nj.nec.com/InformationRetrieval/
37. www.cs.colorado.edu/~summer/cs3202/final-reports/brendan.html
38. www.cs.umbc.edu/~jliu/ir.shtml
39. www.slis.indiana.edu/Research/ir.html
40. www.eurekalert.org/links/Journals_public.html
41. www.digitalcentury.com/encyclo/update/comp_hd.html
42. www.cs.reading.ac.uk/people/dwc/ai.html
43. www.cs.berkeley.edu/~russell/ai.html
44. www.yahoo.com/Science/Computer_Science/Artificial_Intelligence/
45. www.kantrowitz.com/kantrowitz/mark.html
46. www.sil.org/linguistics/computing.html
47. www.sims.berkeley.edu/research/overview.html
48. www.searchtools.com/related/info-retrieval.html
49. www-cse.ucsd.edu/users/rik/MLIA.html
50. www.library.ucsb.edu/untangle/larger.html
51. www.cs.princeton.edu/~aslp/www_resources.html
52. www.cs.mu.oz.au/~leon/agentlinks.html
53. www.cs.rice.edu/~siruguri/gen_pages/docs/bookmarks.html
54. www-ils.unc.edu/iris/sbookmark.html
55. www.cs.umbc.edu/abir/bibliography.shtml
56. www.research.microsoft.com/research/ui/persona/related.html
57. www-isl.stanford.edu/~gray/iii.html
58. www.webreference.com/internet/software/agents.html
59. lieber.www.media.mit.edu/people/lieber/
60. www.cs.cmu.edu/afs/cs.cmu.edu/project/oz/web/related-work.html
61. www.yahoo.com/Science/Computer_Science/Human_Computer_Interaction_HCI_/
62. lieber.www.media.mit.edu/people/lieber/Hotlist.html
63. info.webcrawler.com/mailing-lists/robots/0569.html
64. www.research.microsoft.com/research/ui/persona/isbister.html
65. www.ai.univie.ac.at/oefai/ml/ml-resources.html
66. www.almaden.ibm.com/cs/quest/PUBS.html
67. industry.ebi.ac.uk/~brazma/dm.html
68. www.digimine.com/usama/datamine/specials.html
69. www.cs.bham.ac.uk/~amw/local.html
70. www.ai.univie.ac.at/imkai/lv/winf-ai/lv-ws.html
71. www.princeton.edu/~stengel/MAE345.html
72. www.aaai.org/Pathfinder/html/agri.html
73. www.clark.net/pub/pribut/chess.html
74. www.ai.uniview.ac.at/~juffi/links.html
75. akala.terrashare.com/pricingcentral.html
76. www.aic.nrl.navy.mil/~aha/research/machine-learning.html
77. www.cs.wisc.edu/~shavlik/cs760.html
78. www.yahoo.com/Science/Computer_Science/Artificial_Intelligence/Natural_Language_Processing/
79. www.abacom.com/innomagi/online/computing/ai.html
80. webopedia.internet.com/TERM/n/natural_language.html
81. www.canis.uiuc.edu/~bgross/dl/
82. hul.harvard.edu/ldi/html/metadata.html
83. www.nla.gov.au/padi/format/org.html
84. www.oasis-open.org/cover/gen-apps.html
85. www.sics.se/~lalle/bookmarks.html
86. www.dcc.ufmg.br/~loureiro/pesquisa.html
87. www.doi.org/handbook_2000/bibliography.html
88. web.simmons.edu/~schwartz/mydigital2.html
89. www.codata.org/codata/data_access/wiporefs.html
90. www.cerias.purdue.edu/coast/hotlist/law/
91. lists.essential.org/1996/info-policy-notes/msg00048.html
92. www.dlib.org/dlib/january99/01clips.html
93. www.dmoz.org/Computers/Software/Information_Retrieval/
94. www.w3.org/PICS/
95. www.telnet.edu/jumpstart/parents.html
96. www.ifla.org/II/metadata.html
97. www.research.att.com/projects/tech4kids/t4k.html
98. www.sciam.com/0397issue/0397intro.html
99. www.tispa.org.au/kinnaman/filtering.html
100. www.efa.org.au/News/issue4_1.html
101. www.ukoln.ac.uk/web-focus/w3c/briefings/brief-feb-1998.html

9

TEMPORAL AND SPATIAL DATA MINING

9.1 INTRODUCTION

Many applications maintain temporal and spatial features in their databases; these features cannot be treated as any other attributes and need special attention. To put it in another way, so far we have been asking ourselves 'what' knowledge is being mined, but finding 'when' and 'where' knowledge are also equally important, and these cannot be trivially handled by the methods discussed thus far. In this chapter, we shall study the mining techniques of temporal data and spatial data. The widespread use of GIS by local and federal governments and other institutions, necessitate the development of adequate mining tools for geo-referenced data. Perhaps, the second generation of data mining techniques would contribute in a major way to spatial and temporal data mining.

The first part of this chapter deals with temporal mining techniques. Here, we begin with the basic concepts of temporal mining and why special-purpose techniques are needed for this problem. Then, we proceed to discuss some of the major techniques. It may be mentioned here that we discuss the mining of *sequence data* and *time-series* data in this chapter. Though these topics are relevant in the context of temporal data, they are also important in other applications such as DNA sequencing. The second part of this chapter deals with spatial data mining. The motivation for the special-purpose algorithms for spatial data mining is almost the same as that of temporal mining. Some of the techniques of temporal mining can be trivially extended to spatial data. There are also certain similar mining tasks in these approaches. To avoid repetition, we concentrate only on the techniques specific to spatial data in the second part of this chapter.

In Section 9.2, we shall outline different types of temporal data and inferences. Section 9.3 briefly discusses the features of temporal association rules. In Section 9.4, we give an account of sequence mining techniques. GSP algorithms for sequence mining are discussed in Section 9.5. Other algorithms for sequence mining are outlined in Section 9.6. One of the major temporal data mining techniques is episode mining; Section 9.7 deals with this problem. The event prediction problem, which is a different approach to that of sequence mining and episode mining, is discussed in Section 9.8. Section 9.9 is concerned with time-series analysis for data mining applications. Another related data mining problem is Spatial Data Mining. We discuss the techniques of SDM in the remaining sections of this chapter.

9.2 What is Temporal Data Mining?

By taking into account the time aspect of data in mining techniques, we gain some insight into the temporal arrangement of events and thus an ability to discover cause and effect. *Temporal Data Mining* is an important extension of data mining and it can be defined as the non-trivial extraction of implicit, potentially useful and previously unrecorded information with an implicit or explicit temporal content, from large quantities of data. It has the capability to infer causal and temporal proximity relationships, and this is something that non-temporal data mining cannot do. It may be noted that data mining from temporal data is not temporal data mining, if the temporal component is either ignored or treated as a simple numerical attribute. Also note that temporal rules cannot be mined from a database which is free of temporal components by traditional (non-temporal) data mining techniques. Thus, the underlying database must be a temporal one and specific temporal data mining techniques are also necessary.

Consider, for example, an association rule which looks like—"Any person who buys a car also buys steering lock". But if we take the temporal aspect into

consideration, this rule would be—"Any person who buys a car also buys a steering lock *after* that". Another temporal rule could be—"Flooding in the east coast occurs only *during* the monsoon". A static association rule algorithm would be able to induce that "East-coast flooding occurs only with the monsoon". The concepts of *during* and *after* are explicitly temporal. Take another example: "Data Mining is becoming increasingly popular". It shows another technique of mining from previously mined rules, to find trends in rule sets. Some other examples of temporal knowledge discovery are—"A drop in atmospheric pressure precedes rainfall in 60% of the cases"; "The sequence {semester examination, grading, result processing} occurs every semester"; "Some patients tend to develop reactions after two months with this combination of drugs".

Thus, temporal data mining aims at mining new and hitherto unknown knowledge, which takes into account the temporal aspects of the data. Let us first concentrate on the different temporal aspects of the data.

Types of Temporal Data

When do we say that data contains temporal features? There can be four different levels of temporality.

Static

Static data are free of any temporal reference and the inferences that can be derived from this data are also free of any temporality.

Sequences (Ordered sequences of events)

In this category of data, though there may not be any explicit reference to time, there exists a sort of qualitative temporal relationship between data items. The market–basket transaction is a good example of this category. The entry-sequence of transactions automatically incorporates a sort of temporality. If a transaction appears in the database before another transaction, it implies that the former transaction has occurred before the latter. There may not be any reference to quantitative temporal relationships. While most collections are often limited to the sequence relationships *before* and *after*, this category also includes the richer relationships, such as *during*, *meet*, *overlap*, etc. Such relationships are called qualitative relationships between time events. Sequence mining is one of the major activities in temporal data mining.

Timestamped

In this category the temporal information is explicit. Note that the relationship can be quantitative, in the sense that we can not only say that one transaction occurred before another, but also the exact temporal distance between the data elements. Some examples include census data, land-use data and satellite meteorological data. The inference made here can be temporal or non-temporal. Time series data are a special

case of this category, with the events being uniformly spaced on the time scale. Time series data mining is another topic that we shall be discussing in this chapter.

Fully Temporal

In this category, the validity of the data element is time-dependent. The inferences are necessarily temporal in such cases.

Temporal Data Mining Tasks

Some of the conventional mining tasks can be extended with some additional temporal information as described below.

Temporal Association

The association rule discovery can be extended to temporal association. In static association rule discovery tasks, we were trying to find static associations between two non-temporal itemsets. In the temporal association discovery, we attempt to discover temporal association between non-temporal itemsets. We can say that: "70% of the readers who buy a DBMS book also buy a Data Mining book *after* a semester".

Temporal Classification

We can cluster the data items along temporal dimensions. For example, we can identify a set of people who go for a walk in the evening and a set of people who go for a walk in the morning. We can categorize sets of patients based on their visit sequence to different medical experts. We can also categorize sets of net surfers based on the mouse-click sequence.

Temporal Characterization

An interesting experiment would be to extend the concept of decision tree construction on temporal attributes. For example, a rule could be: "The first case of filaria is normally reported after the first pre-monsoon rain and during the months of May–August".

Trend Analysis

The analysis of one or more time series of continuous data may show similar trends, i.e., similar shapes across the time axis. For example, "The deployment of the Data Mining system is increasingly becoming popular in the banking industry". This type of analyses are of a higher level than the earlier ones. Here, we are trying to find the relationships of change in one or more static attributes, with respect to changes in the temporal attributes.

Sequence Analysis

Events occurring at different points in time may be related by causal relationships, in that an earlier event may appear to cause a later one. To discover such relationships,

sequences of events must be analyzed to discover common patterns. This category includes the discovery of frequent events and also the problem of event prediction. It may be noted that frequent sequence mining finds the frequent subsequences; while event prediction predicts the occurrences of events which are rare.

In the following sections, we shall elaborate upon some of the techniques for the tasks discussed above.

9.3 Temporal Association Rules

Association rules identify whether a particular subset of items are supported by an adequate number of transactions. They are normally static in character. As we have mentioned earlier, a static association rule discovers the association between any two events—for instance, the purchase of aerated soft drinks and stomach upsets. However, the association may not indicate any causal relationship, unless the temporality in the association is brought out. One can extend the association rule discovery to incorporate temporal aspects too. It should be noted that the presence of a temporal association rule may suggest a number of interpretations, such as

- The earlier event plays some role in causing the later event.
- There is a third set of reasons that causes both events.
- The confluence of events is coincidental.

Temporal association rules are sometimes viewed in the literature as *causal rules*. Causal rules describe relationships, where changes in one event cause subsequent changes in other parts of the domain. They are common targets of scientific investigation within the medical domain, where the search for factors that may cause or aggravate particular disease is important. The static properties, such as gender, and the temporal properties, such as medical treatments, are taken into account during mining.

While the concept of association rule discovery is the same for temporal and non-temporal rules, algorithms designed for conventional rules cannot be directly applied to extract temporal rules. The reason is that classical association rules have no notion of order, whilst time implies an ordering.

9.4 Sequence Mining

An efficient approach to mining causal relations is sequence mining. As observed earlier, sequence mining is a topic in its own right and many application domains such as DNA sequence, signal processing, and speech analysis require mining of sequence data, even though there is no explicit temporality in the data. Discovering sequential

patterns from a large database of sequences has been recognized as an important problem in the field of knowledge discovery and data mining. To put it briefly, given a set of data sequences, the problem is to discover subsequences that are frequent, in the sense that the percentage of data sequences containing them exceeds a user-specified minimum support.

Mining frequent sequential patterns has found a host of potential application domains, including retailing (i.e., market–basket data), telecommunications and, more recently, the World Wide Web (WWW). In market–basket databases, each data sequence corresponds to items bought by an individual customer over time, and frequent patterns can be useful for predicting future customer behaviour. In telecommunications, frequent sequences of alarms output by network switches capture important relationships between alarm signals that can then be employed for on-line prediction, analysis, and correction of network faults. In the context of the WWW, server sites typically generate huge volumes of daily log data capturing the sequences of page accesses for thousands or millions of users. Let us begin with formally defining the sequence mining problem.

Sequence Mining Problem

The most general form of the sequence mining problem [Zaki, 1998] can be stated as follows:

Let $\Sigma = \{i_1, i_2, \ldots, i_m\}$ be a set of m distinct items comprising the alphabet.

An event is a non-empty, disordered collection of items. Without any loss of generality, we write the items in an event in some predefined order. An event is denoted as $(i_1, i_2, \ldots, i_k)$, where i_j is an item in Σ. Often, we drop the ',' and parentheses for notational convenience.

Any event that is given as input will also be called a transaction. Thus, transactions and events have the same structure, except that a transaction is known to us prior to the process and an event is generated during the algorithm. We use both terms interchangeably if there is no ambiguity.

Definition 9.1 Sequence

A *sequence* is an ordered list of events. A sequence, α, is denoted as $(\alpha_1 \rightarrow \alpha_2 \rightarrow \ldots \rightarrow \alpha_q)$, where α_i is an event. A sequence is called a k-sequence, if the sum of the cardinalities of α_i is k.

A subsequence is a sequence within a sequence, preserving the order. In other words, its items need not be adjacent in time but their ordering in a sequence should not violate the time ordering of the supporting events. A subsequence can be obtained from a sequence by deleting some items and/or events. A formal definition of a subsequence is given below.

DEFINITION 9.2 SUBSEQUENCE

The sequence $s = (\alpha_1 \rightarrow \alpha_2 \rightarrow \ldots \rightarrow \alpha_q)$ is said to be a subsequence of $s^1 = (\beta_1 \rightarrow \beta_2 \rightarrow \ldots \rightarrow \beta_r)$, if there exist indices $t1 < t2 < \ldots < tq$ of s^1, such that $\alpha_1 \subseteq \beta_{t1}$, $\alpha_2 \subseteq \beta_{t2}$, $\ldots$, $\alpha_q \subseteq \beta_{tq}$,

A sequence s^1 is said to *support* another sequence s if s is a subsequence of s^1. Let **D** be the database of input sequences and a sequence is a set of temporally-ordered transactions.

DEFINITION 9.3 FREQUENCY

The frequency of a sequence s, with respect to this database **D**, is the total number of input sequences in **D** that support it.

DEFINITION 9.4 FREQUENT SEQUENCE

A frequent sequence is a sequence whose frequency exceeds some user-specified threshold. A frequent set is *maximal* if it is not a subsequence of another frequent sequence.

The rationale behind frequent sequences lies in detecting precedence and causal relationships that make them statistically remarkable.

EXAMPLE 9.1

Let us consider the following transaction database (Table 9.1). There are six items,

Table 9.1 A Sample of Sequence Data

Customer	A	B	C	D	E	F
C1	1	1	0	1	1	1
C2	0	0	1	1	0	1
C1	0	1	1	0	1	1
C3	0	0	0	1	1	1
C2	1	1	1	1	0	1
C1	0	1	0	0	1	0
C3	1	0	1	1	1	0
C2	0	1	0	0	1	0
C4	1	1	1	1	1	1
C1	0	0	1	1	1	1
C4	0	1	0	0	0	1
C3	1	0	1	1	1	1
C2	0	1	1	0	0	0
C4	1	0	1	1	1	1
C2	0	1	0	0	0	0

Table 9.2 One Sequence from the Sequence Data of Table 9.1

Customer	A	B	C	D	E	F
C1	1	1	0	1	1	1
C1	0	1	1	0	1	1
C1	0	1	0	0	1	0
C1	0	0	1	1	1	1

that is, $\Sigma=\{A, B, C, D, E, F\}$, and the database depicts the purchases made by 4 customers during a certain period of time. The transactions are ordered chronologically. Table 9.2 extracts one sequence.

We write the transaction-sequences of all customers in the following form, thereby indicating the items a transaction contains.

Sequence 1: (A, B, D, E, F) → (B, C, E, F) → (B, E) → (C, D, E, F)
Sequence 2: (C, D, F) → (A, B, C, D, F) → (B, E) → (B, C) → (B)
Sequence 3: (D, E, F) → (A, C, D, E) → (A, C, D, E, F)
Sequence 4: (A, B, C, D, E, F) → (B, F) → (A, C, D, E, F)

Thus, the database of sequences **D** consists of four sequences.

Please note that AC (it is, in fact, an abbreviation of (A, C)) is not a subsequence of Sequence 1, but it is a subsequence of Sequence 3 and also of Sequence 4. But the sequence A → C is a subsequence of Sequence 1, whereas it is not a subsequence of Sequence 3. By AC, we mean that there should be a transaction containing both A and C. By A → C, we mean that there is a transaction containing C which appears after – not necessarily, immediately after – another (different) transaction containing A.

The frequency of AC in **D** is 3. Note that we do not count multiple occurrences of AC in the same sequence. The total number of sequences that support AC is only three, namely Sequence 2, Sequence 3 and Sequence 4. Similarly, the frequency of B → D is 2. The sequences supporting B → D are Sequence 1 and Sequence 4.

Note that B → BE is not supported by Sequence 4, which supports BE → B. The subsequence B → BE indicates that a transaction containing B and E follows (sometime later) another transaction containing B. On the other hand, BE → B represents a subsequence of transactions, in which a transaction containing B and E occurs before a transaction containing B. Both are 3-sequences.

Normally, a simple sequence mining problem is concerned with the temporal order of the events within a sequence of transactions. A more general problem is when we also focus on the temporal distance between the events. That is, A → C should indicate the temporal gap between the transaction containing A and the transaction containing C. It is relevant when the correlation between the two events ceases to be effective after some period of time. For example, when we are trying to find out the

causal relationship between consuming a beverage and having a stomach upset, it will be irrelevant to correlate two such events occurring within a gap of one year of each other. Thus, we can specify the time distance in terms of a distance threshold, **d**. So, B $\rightarrow_{\mathbf{d}}$ BE denotes that the event containing B and E occurs in not more than **d** transactions after the transaction containing B.

For example, there is no sequence which supports ABD $\rightarrow_1$ D, whereas ABD $\rightarrow_2$ D is supported by Sequence 4. When **d** =1, we say that the problem is a *contiguous sequence mining problem.*

A simple sequence mining problem is the sequence mining problem where each transaction contains a single item. There are many applications in which simple sequence mining is relevant.

9.5 The GSP Algorithm

The algorithms for solving sequence mining problems are mostly based on the *A priori* (level-wise) algorithm. One way to use the level-wise paradigm is to first discover all the frequent items in a level-wise fashion. It simply means counting the occurrences of all singleton elements in the database. Then, the transactions are filtered by removing the non-frequent items. At the end of this step, each transaction is a modified transaction consisting of only the frequent elements it contains. We use this modified database as an input to the GSP algorithm. This process requires one pass over the whole database.

The GSP Algorithm

GSP makes multiple passes over the database. In the first pass, all single items (1-sequences) are counted. From the frequent items, a set of candidate 2-sequences are formed, and another pass is made to gather their support. The frequent 2-sequences are used to generate the candidate 3-sequences, and this process is repeated until no more frequent sequences are found. There are two main steps in the algorithm.

Candidate Generation

Given the set of frequent (k–1)-frequent sequences $F_{(k-1)}$, the candidates for the next pass are generated by joining $F_{(k-1)}$ with itself. A pruning phase eliminates any sequence, at least one of whose subsequences is not frequent.

Support Counting

Normally, a hash tree-based search is employed for efficient support counting. Finally non-maximal frequent sequences are removed.

GSP Algorithm

F_1 = the set of frequent 1-sequence
$k = 2$,
do while $F_{k-1} \neq \varnothing$;
    generate candidate sets C_k (Set of candidate k-sequences);
    for all input sequence s in the database ***D do***
    increment count of all a in C_k if s supports a
    $F_k = \{a \in C_k$ such that its frequency exceeds the threshold$\}$
$k = k + 1$
set of all frequent sequences is the union of all F_ks
end do.

The above algorithm looks like the *a priori* algorithm. One main difference is however the generation of candidate sets. Let us assume that A → B and A → C are two frequent 2-sequences. The items involved in these sequences are (A, B) and (A, C), respectively. The candidate generation in the usual *a priori* style would give (A, B, C) as a 3-itemset, but in the present context we get the following 3-sequences as a result of joining the above 2-sequences

A → B →C , A → C → B, and A → BC.

The candidate-generation phase takes this into account.

The GSP algorithm discovers frequent sequences, allowing for time constraints such as maximum gap and minimum gap, among the sequence elements. Moreover, it supports the notion of a sliding window, i.e., of a time interval within which items are observed as belonging to the same event, even if they originate from different events.

9.6 SPADE

SPADE is another algorithm based on the level-wise paradigm. In the year 2000, Zaki proposed **SPADE** (**S**equential **PA**ttern **D**iscovery using **E**quivalence classes) for discovering the set of all frequent sequences. SPADE draws its motivation from the level-wise algorithm (GSP too) and vertical mining. Initially, SPADE uses a vertically structured database (recall VIPER and FP-Tree in Chapter 4). In this structure, a list is associated with each item. This list include the *sequence-id* and the *transaction-id* containing the item. Using this structure, all frequent sequences can be enumerated via simple temporal joins (or intersections) on id-lists. The vertical structure for the sequence database is illustrated below. For every item, we have a list containing the sequence-id and the transaction id containing the item.

EXAMPLE 9.2

For the example given in Example 9.1 the vertical structure can be written as follows

A:
(Seq1, Trans1), (Seq2, Trans2), (Seq3, Trans2), (Seq3, Trans3), (Seq4, Trans1), (Seq4, Trans3).

B:
(Seq1, Trans1), (Seq1, Trans2), (Seq1, Trans3), (Seq2, Trans2), (Seq2, Trans3), (Seq2, Trans4), (Seq2, Trans5), (Seq4, Trans1), (Seq4, Trans2).

C:
(Seq1, Trans2), (Seq1, Trans4), (Seq2, Trans1), (Seq2, Trans2), (Seq2, Trans4), (Seq3, Trans2), (Seq3, Trans3), (Seq4, Trans1), (Seq4, Trans3).

D:
(Seq1, Trans1), (Seq1, Trans4), (Seq2, Trans1), (Seq2, Trans2), (Seq3, Trans1), (Seq3, Trans2), (Seq3, Trans3), (Seq4, Trans1), (Seq4, Trans3).

E:
(Seq1, Trans1), (Seq1, Trans2), (Seq1, Trans3), (Seq1, Trans4), (Seq2, Trans3), (Seq3, Trans1), (Seq3, Trans2), (Seq3, Trans3), (Seq4, Trans1), (Seq4, Trans3).

F:
(Seq1, Trans1), (Seq1, Trans2), (Seq1, Trans4), (Seq2, Trans1), (Seq2, Trans2), (Seq3, Trans1), (Seq3, Trans3), (Seq4, Trans1), (Seq4, Trans2), (Seq4, Trans3).

It is not important to have exactly the same list as shown above to represent the vertical structure. The main idea is to store for each item the set of pairs—(*sequence-id, transaction-id*). The actual implementation of this structure can be different. The vertical structure is very useful to generate candidate sets in a level-wise fashion. For example, we can generate a similar structure for $A \rightarrow B$, by checking the transaction ids of occurrences of A and B. The corresponding structure is as follows:

$A \rightarrow B$
(Seq1, Trans1) → (Seq1, Trans2), (Seq1, Trans3),
(Seq2, Trans2) → (Seq2, Trans3), (Seq2, Trans4), (Seq2, Trans5),
(Seq4, Trans1) → (Seq4, Trans2).

The core of SPADE is to employ a level-wise paradigm (similar to GSP) with the vertical structure and the temporal join, using the vertical structure. However, it is space-consuming to carry the vertical structure along the levels for the candidate sets. Note that though GSP requires to store only the frequency of $A \rightarrow B$ as 3, SPADE maintains the structure shown above for this 2-sequence. SPADE surmounts this difficulty by decomposing the problem into smaller subproblems.

SPADE exploits the lattice structure of the set of all subsequences to decompose the problem into smaller subproblems. We have discussed the lattice structure of itemsets in the context of the association rule discovery in Chapter 4. The nicety of the lattice structure is not preserved when we consider the set of sequences in place of the set of items. The main difficulty is that there is no unique k-sequence resulting from the join of two $(k-1)$-sequences (on the other hand, we can get a unique k-itemset as a result of the join of two $(k-1)$-itemsets). We have noticed this phenomenon above. By relaxing the uniqueness of the join and allowing multiple candidates as the result of the join of two subsequences, we can get a *hyper-lattice* structure of the set of sequences. In principle, the hyper-lattice is unbounded at the top. The item A can generate an infinite sequence A→A→.... A→... But when the transactions are of finite length and the set of transactions in a sequence is bounded from above, the hyper-lattice structure is also bounded.

Thus, SPADE starts from the bottom-most (the least element) of the lattice and gradually works in a bottom-up way to generate all frequent sequences. It maintains the vertical structure as it proceeds from the least element to the maximal elements.

In order to optimize the memory requirement and efficiency, SPADE decomposes the original search space (lattice) into smaller subspaces (sublattices), which can be processed independently in the main memory. The key idea of decomposition is a prefix-based equivalence relation. Consider a sequence D → BF → A and sequence D → BF → E. Note that although the sequences are distinct they share the same prefix, namely D → BF. DB → F → E and DBFC also share the same prefix. Similarly, the sequence D → BF → A and D → CE → A also share the prefix D. However, the length of common prefixes in the above two cases are different. It is easy to see that for a fixed value of the length, say k, an equivalence relation can be defined as q_k and two sequences are said to be equivalent with respect to this relation, if they share a common prefix of length k. The equivalence relation q_k induces a partition on the set of sequences (having a lattice structure) and each equivalent class is a sublattice in this structure. Each of these sublattices can be handled separately for the join operation. However, the pruning stage requires interaction among the sublattices.

In the above example, different equivalent classes of the equivalent relation q_1 are [A], [B], [C], [D], [E], and [F], where [X] denotes all the frequent sequences which have X as the first prefix. We can first independently solve the sequence mining subproblems [A], ..., [F] separately and by examining the interactions between the equivalent classes in the hyper-lattice, we can generate all the frequent sequences. The algorithm proceeds recursively. In order to solve [A], we may have to solve [AB], [A → B], etc. In practice, one need not go beyond two to three levels deep in the recursion process. In most cases, the size of the subproblem rapidly decreases and it becomes small enough to be handled in the main memory. SPADE begins from the bottom (least, $k = 0$) elements of this lattice and moves upwards either in a breadth-first manner or a depth-first manner. This bottom-up movement is carried out by

incrementing k by 1 at every step (level-wise philosophy), and at every iteration pruning takes place. SPADE normally requires three database scans, or only a single scan with some preprocessed information. SPADE appears to be an efficient method for frequent sequence mining.

9.7 SPIRIT

Garofarakasi, Rajeev Rastogi and K Shim, in 1999, observed that most of the earlier algorithms lack an user-controlled focus. The algorithms can be considered as an 'unfocused' method. Typically, the user can only specify the support and confidence bounds and the system returns a very large number of sequential patterns. Most of the frequent patterns, so discovered, are hardly of any interest to the user. The *constrained sequence mining problem* allows the user to specify a set of constraints, so that only the set of frequent sequences that satisfy the constraints are returned. The question now is whether the user can specify his interest in some form. SPIRIT (Sequential Pattern Mining with Regular Expression Constraints) is an sequence mining algorithm that allows the user to specify the constraints in the form of regular expressions.

Constrained sequence mining problem is used to discover frequent sequences from **D**, such that each of these frequent subsequences satisfy the user-specified constraint set **C**. Such a problem requires two additional tools: 1) The user should be provided with a reasonably expressive language to express his constraint set **C**, and 2) The constraint set **C** should be properly blended within the frequent sequence discovery process. It becomes trivial and uninteresting to first discover all the frequent sequences and then check which of these satisfy the user-specified constraints **C**, so as to return only the required frequent patterns. Therefore, it is desirable that any constrained sequence mining algorithm must push the constraint-checking stage inside the candidate-generation step. But this is not easy. The level-wise paradigm cannot be trivially extended, as the downward closure (anti-monotone) property may be affected.

Let us assume that **C** possesses the downward-closure property. This means that for any subsequence that satisfies **C**, all its subsequences also satisfy **C**. In that case, we generate candidate sequences at level k from the subsequences of F_{k-1} and prune the k-sequences that do not satisfy **C**. In the pruning phase, we must also make sure that no subsequence of the candidate k-sequence is infrequent. Thus, the constraint checking can be done at the candidate-generation phase.

However, if **C** does not possess the downward-closure (or, anti-monotone) property, then the method of embedding constraint checking within the candidate generation process will not work. To see this, let us assume that **C** is so defined that $A \rightarrow B \rightarrow C$ satisfies **C**, but none of the frequent 2-sequences $A \rightarrow B$ and $A \rightarrow C$ satisfies **C**. In that case, F_{k-1} does not contain the frequent 2-sequences that can generate the 3-sequence $A \rightarrow B \rightarrow C$. In such cases, the brute force method would be inevitable to generate all

unconstrained frequent sequences using GSP or SPADE and then retain only those frequent sequences which satisfy the constraint. There can also be a middle way. We can relax the constraint **C** to an even weaker constraint **C′**, such that **C′** exhibits the downward-closure property. Then, we can embed **C′** within the algorithm. In this process, the number of candidates would be much less, because we will be generating only those candidates which satisfy the weaker constraints. We will still have to check at the end, the original constraint **C**. Depending upon the closeness of **C′** to **C**, the candidate-generation step can be optimized.

SPIRIT is an algorithm (a family of algorithms, to be precise) to solve constrained sequence mining problems with simple sequences. In other words, SPIRIT focuses on sets of transactions having only single items. SPIRIT adopts regular expressions as the language to express **C** and attempts to embed these constraints within the mining process, to gain efficiency in mining. The major advantages of regular expressions as constraints are that they are sufficiently expressive in the context of sequence mining and it is equivalent to a deterministic finite automaton. In one sense, SPIRIT is a family of algorithms which differ among themselves in what extent the regular expression constraints are pushed into the mining process—how weak is the constraint relaxation. In fact, different levels of relaxations give rise to different types of algorithms within the SPIRIT family. The equivalence of the regular expression and the finite automata comes in handy here in the relaxation process. Since **C** is in the form of the regular expression, we can construct a finite automaton that accepts only those sequences that satisfy **C**. Acceptance by a finite automaton means that the items in the sequence admit a sequence of state transitions leading to any of the final (accept) states from the start state. The relaxation of **C** can be done in two ways:

- We can allow the sequences which admit a sequence of state transitions leading to a final state from any state (not necessarily from the start state).
- We can allow the sequences which admit state transition from any one state to any other state.

Note that these relaxations exhibit the downward-closure property to some extent. Hence, such relaxed constraints can be embedded within the candidate set generation stage. However, after completion of the level-wise algorithm, we have a final step of checking the original constraint to see which of the frequent sequences generated using relaxed constraints satisfy the original set of constraints. The main advantage of pushing relaxed constraints within the algorithm is that we have an optimal set of candidate sequences at every level for the frequency count.

9.8 WUM

WUM (**W**eb **U**tilization **M**iner) is yet another constrained-sequence miner. It makes

use of a language, MINT, for the user to specify the appropriate constraints for the application. Its primary purpose is to analyze the navigational behaviour of users on the web, but it is appropriate for sequential pattern discovery in any type of log. In MINT, frequent sequence discovery corresponds to the specification of a lower-bound support threshold. In exactly the same way, the user can specify an upper-bound support threshold to restrict the search space to that of rare sequences. The main idea is to apply mining on disk-resident condensed data instead of the original log of events, but on a condensed disk-resident tree. This tree is built incrementally by a backend service that extracts sequences from the original log and adds them onto the tree, merging common sequence prefixes (recall the Prefix Tree in Chapter 4). The number of merged prefixes corresponding to each tree node is retained in the node's statistics and is used to compute support values during mining. The miner generates candidate sequences while traversing the tree, using heuristics to reject subsequences as soon as possible. By processing a relatively small preaggregated data structure, the execution overhead is reduced without large space tradeoffs.

9.9 Episode Discovery

Another important temporal data mining problem is the discovery of episodes that occur frequently within sequences. Heiki Mannila and his team formulated and devised algorithms for the discovery of frequent episodes. Zaki's formulation of sequence mining, given above, is general enough to have episode mining problem as its special case. Keeping the above definition in mind, we shall now formulate the episode discovery problem. Episode discovery is similar to sequence mining, but for the following special assumptions:

- The input sequence is a single long input sequence, unlike in the case of sequence mining where we have a set of data sequences.
- The events (in this context, referring to a transaction as an event is more appropriate) are typically single item events.
- An episode is a subsequence.

The frequent episode discovery problem is to find all episodes that occur frequently in the event sequence within a time window. Let us define some basic concepts that are necessary in the present context.

Definition 9.5 Event

An event is a pair $\{A, t\}$, where A is an single item event, and t is an integer timestamp of the occurrence of A.

DEFINITION 9.6 EVENT SEQUENCE

An event sequence *Ev_Seq* is a triplet *(Seq, T_start, T_end),* with *T_start* and *T_end* denoting the start and end time of the sequence; and $Seq = (\{A_1, t_1\}, \{A_2, t_2\}, \ldots \{A_k, t_k\})$ is an ordered sequence of events.

DEFINITION 9.7 TIME WINDOW

A *time window* W for *Ev_Seq(Seq, T_start, T_end),* is an event sequence (W, t_s, t_e), where $t_s \geq T\_start$ and $t_e \leq T\_end$ and $(t_e - t_s)$ is said to be width of the window.

We shall represent the data in the form of a graph, where each event corresponds to a node. The precedence relationships among nodes represent the temporal precedence among events. Given a set of nodes and a set of events, it is necessary to specify the mapping to identify the correspondence between nodes and events.

DEFINITION 9.8 EPISODE

An episode a is a triple $(V, \leq, g)$, where V is a set of nodes, $\leq$ is a partial order on V, and g: $V \rightarrow I$ is a mapping associating each node with an event satisfying the partial order.

DEFINITION 9.9 PARALLEL AND SERIAL EPISODES

If the partial order $\leq$ is a trivial partial order, the episode is called a *parallel episode.* If the partial order is a total ordering, then the episode is called *serial episode.*

An epsidoe $A \rightarrow B \rightarrow C$ is a serial episode. A serial episode occurs in a given sequence only if A, B and C occur in this order relatively close. There can be other events occurring between these three. In a parallel episode, there is no constraints on the relative order of the events. In general, we can have an episode which has some partial constraints on order too.

DEFINITION 9.10 SUBEPISODE

An episode is to be a subepisode if it is obtained by deleting some events from an episode.

DEFINITION 9.11 OCCURRENCE OF AN EPISODE IN AN EVENT SEQUENCE

An episode is said to be occurring in a sequence if the events corresponding to the nodes of the episode appear in the sequencing, preserving the partial order of the episode.

DEFINITION 9.12 FREQUENCY OF AN EPISODE

The frequency of an episode, with respect to a given window width in a sequence, is

the fraction of all the windows in the sequence of the specified width in which the episode occurs. Let us assume that $W(\omega)$ is the set of total number of time windows of width ω in the given sequence, and $W(\omega, \alpha)$ is the set of windows in $W(\omega)$ in which the episode α occurs. Then, the frequency is the ratio of $W(\omega, \alpha)$ to $W(\omega)$.

A frequent episode is the episode that has a frequency above a user-specified threshold.

The episode discovery problem can be stated as follows. Given a sequence *Ev_seq*, a class C of episodes, a window width ω, and a frequency threshold σ, it is to find all frequent episodes with respect to ω and σ in *Ev_seq*.

EXAMPLE 9.3

Consider the following sequence (Figure 9.1) on a time interval (0, 50).

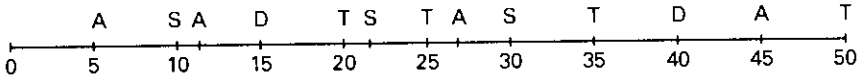


Figure 9.1 An example of a sequence

We define *Evt_Seq* as (*Seq*, 0, 50) and it consists of

(*A*, 5), (*S*, 10), (*A*, 12), (*D*, 15), (*T*, 20), (*S*, 22), ...

The following episodes are occurring in the sequence.

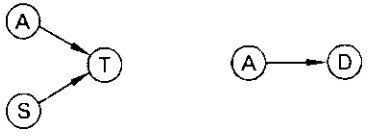


Figure 9.2 Serial and Parallel episodes

If we take the window width $\omega = 10$, then $A \rightarrow D$ occurs in only two windows of width 10 in the given sequence.

EPISODE DISCOVERY PROCESS

Like many other mining problems, the discovery of a frequent episode is also influenced by the level-wise algorithm. The algorithm makes multiple passes and the candidate sets of $(k+1)$-episodes are generated from frequent k-episodes. This is done by finding the pair of frequent k-episodes having a common prefix of size $k-1$ and preserving the partial ordering.

It makes a database pass to count the frequency. Mannila and Toivonen distinguish between serial and parallel episodes, as well as between simple episodes (those

containing no binary predicates) and non-simple episodes. They propose an algorithm based on the iterative construction of simple frequent episodes from simple frequent subepisodes. They further extended their framework to discover generalized episodes, which allow one to express arbitrary unary conditions on individual sequence events, or binary conditions on event pairs. In order to count the frequency of occurrences of an episode, it is necessary to recognise an episode within a given window. Two adjacent windows $W_1=(W_1, t_s, t_e)$ and $W_1=(W_1, t_{s+1}, t_{e+1})$ are typically very similar to each other. Thus after determining the set of episodes present in W_1, we can make incremental updates to achieve the *slide* of the window.

9.10 Event Prediction Problem

The discovery of frequent sequences is inappropriate for many applications where sequence pattern discovery is relevant. Consider, for instance, error discovery which is an important application domain. Since errors are rare events, the statistical support of such a sequence is low. Hence, if we restrict the search space to frequent sequences, it will be hard to find rare events. On the other hand, if we reduce the threshold above which a sequence is considered frequent, it will be practically impossible to inspect the result and distinguish interesting patterns from any trivial sequence. The problem of predicting failure (or any other event) differs from sequence prediction problems in that the data (i.e., events) are timestamped, and differs from time-series prediction problems in that the data consists mainly of categorical, non-numerical transactions.

Event Prediction Problem

The event prediction problem is to predict a type of future event, the *target* event, based on past events. More specifically, the problem is to find a prediction rule that successfully predicts future target events by taking the input as timestamped records with categorical items. This problem is particularly interesting in situations where the target event occurs infrequently in the event sequence. When we try to predict an event, we also keep in mind the target event's *prediction period*, which means it must occur at least *warning time* time units before the target event and no more than *monitoring time* time units before the target event. It is interesting to note that in event prediction problems a target event is said to be correctly predicted if at least one prediction is made within its prediction period, regardless of any subsequent negative predictions. Thus, the reliability of a positive prediction is not affected by the presence of negative predictions.

Every event is represented by a tuple consisting of a timestamp field and a number of features. We introduce a wildcard in the event as '?'. Each feature in an event is

permitted to take on any of a predefined list of valid feature values, as well as the wildcard ("?") value, which matches any feature value. For example, if the events in a domain are described by three features, then the event <?, ?, b> would match any event in which the third feature has the value "b".

We shall briefly outline the underlying principles of two major algorithms for prediction problems.

PLANMINE

The PlanMine algorithm proposed by Zaki, Lesh and Ogihara deals with the prediction problem in the context of plan failure prediction. PlanMine attempts to identify frequent events that cause plan failures, by filtering out frequent but uninteresting events. When frequent and uninteresting events are removed, the remaining events become more dominant in the data set. The algorithm works as follows.

PlanMine applies the SPADE algorithm which produces an initial set of sequence rules. Then, pruning is applied in the following multiple phases:

- In a first phase, the data set of plans is split into good and failed plans. The 'good' events are separated and 'failed' events are retained. The 'good' events though are not used for prediction, they act as reference in the subsequent pruning stages.
- The next pruning phase removes patterns which correspond to sequences that are frequent in the data set of bad plans and have high support in the data set of good plans.
- The next pruning phase removes the redundant patterns. A sequence is redundant if it contains a subsequence having the same support value as itself for both data sets of good and bad plans.
- Finally, dominated patterns are removed. A pattern is dominated if it contains a subsequence with lower support in good plans and higher support in bad plans as the pattern itself.

TIMEWEAVER

Weiss and Hirsh propose a supervised learning technique to predict rare events in sequences. Their system Timeweaver, based on the genetic algorithm, is designed to predict hardware failures. It solves by generating prediction patterns that forecast the target events (in this case, the failures).

A prediction pattern is a sequence of events subject to ordering and temporal constraints. The GA algorithm is used to generate a population of prediction patterns. Each individual in GA is a prediction pattern. Let us formalize the concept of a prediction pattern.

A *prediction pattern* is a sequence of events in which consecutive events are

connected by an ordering primitive. The following are the ordering primitives and these are used to define ordering constraints between these events:

- the *wildcard* "*". If we say A*D, then any number of events between A and D may be possible. So the prediction pattern A*D matches ABCD .
- the *next* "." This primitive indicates that there is no events occur in between. So the prediction pattern A.B.C only matches ABC.
- the *parallel* "|". This primitive allows events to occur in any order and is commutative so that the prediction pattern A|B|C will match, amongst others, CBA.

While expressing an ordering constraint, we can combine the set of primitives. The "|" primitive has highest precedence. For example, the pattern "A.B*C|D|E" matches an A, followed immediately by a B, followed sometime later by a C, D and E, in any order. Some of the matched patterns are

ABXXCED
ABAAEDC
ABBCDFGREDCAAA.

The basic steps in our steady-state GA are shown below.

Genetic Algorithm

Initialize population
while do stopping criteria not met
    select two individuals from the population
    apply the crossover operator with probability P_C
    and mutation operator with probability P_M
    evaluate the two newly formed individuals
    replace two existing individuals with the new ones
end do

The population is initialized by generating prediction patterns which contain only a single event, where the feature values in this event are set 50% of the time to the wildcard value and the remaining time to a randomly-selected valid feature value.

Another crucial component of GA is the fitness measure. The fitness function must take an individual pattern and return a value indicating how good the pattern is at predicting target events. Conventional quality measures based on accuracy are not appropriate for this notion of event prediction. Two measures are defined, *recall* and *precision*. *Recall* is the percentage of events correctly predicted. *Precision* is the percentage of precisions that are correct. This latter measure is refined in several ways that differ in the treatment of false predictions.

Timeweaver, in order to prevent premature convergence and encourage diversity in

the population, uses *shared fitness* as the basis of the crossover and selection operators. This measure depends on the precision and recall of the population member and on its phenotype distance from other members of the population. After generating a satisfactory initial group of patterns, a set of prediction rules are created in a second pass by selecting those patterns that improve the collective recall of the set without sacrificing precision. Timeweaver has proved to be very efficient.

A prediction pattern matches a sequence of events within an event sequence if:

1. The events within the prediction pattern match events within the event sequence,
2. The ordering constraints are satisfied, and
3. The matched events in the event sequence occur within a period not exceeding the pattern duration.

Once a match succeeds, a target event is predicted.

9.11 Time-series Analysis

Time series are an important class of complex data objects; they arise in many applications. For example, stock price indices, volume of product sales, telecommunication data, one-dimensional medical signals, audio data, and environmental measurement sequences are all time-series databases. Time-series data are sequences of real numbers representing measurements at uniformly-spaced temporal instances. Time-series analysis, like all other forms of data analysis, is used to characterize or explain the reasons for the behaviour of a system and/or to predict its future behaviour.

Time-series analysis is a well-studied subject in statistics and signal processing. We shall focus on the basic time-series analyses in the context of data mining. Note that the most important aspects of time-series data is that these are sequence data but uniformly spaced on the temporal attribute. Analysis tasks of time series include feature extraction of time-series data; computation of similarity measure among time series data set; segmentation of data set; matching two time series data; clustering and classifying time-series data. We shall introduce the related concepts first.

Definition 9.13 ***n*-series**

An n-series X is a sequence $\{x_1, x_2, \ldots, x_n\}$ of real numbers. Each n-series X has an average $\alpha(X)$ and a deviation $\sigma(X)$:

$$\alpha(X) = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \text{and} \quad \sigma(X) = \left[\frac{1}{n}\sum_{i=1}^{n}(x_i - \alpha(X))^2\right]^{\frac{1}{2}}$$

Similarity Function

While mining a time-series database we need an essential primitive operation, called the *similarity function*. It is often necessary to search within a series database for those series that are similar to a given query series. This primitive is needed, for example, for prediction and clustering purposes. For instance, we may be interested in finding the months in five years having similar sales patterns; or in classifying companies based on similar stock price fluctuations. While the statistical literature on time-series is vast, it has not studied similarity notions that would be appropriate for data mining applications. Typically, the task is to define a function Sim(X; Y), where X and Y are two time-series, and the function value represents how "similar" they are to each other.

A simple approach would be to define the similarity functions of X and Y in terms of their Lp-distance as points of R^n. The presence of outliers, which is common in time-series data, makes such schemes unsuitable. Outliers are values that are measurement errors and should be omitted when comparing the sequence against others. A grossly outlying value can cause two otherwise identical series to have a large distance. This similarity measure is not suitable to determine similarity for series in different scales and different shifts.

Scale-free Similarity: Assume that two companies have identical stock price fluctuations, but one's stock is worth twice as much as the others' at all times. Thus, though the fluctuation patterns are similar, the series containing numerical values are not similar. For a company, the sales pattern may be similar throughout the year but the sales volume may be different for different months. It is important to identify such similar time-series objects in data mining applications.

Shift-free Similarity: The temperature on two different days may start at different values, but then go up and down in an exactly similar way. Thus, we have the same series with two different baselines.

For example, consider a time series $z = (20, 20, 21, 21, 20, 20, 23, 23)$ and the series $z' = (43, 43, 45, 45, 43, 43, 49, 49)$. Note that these two series are similar, except that the latter has a different starting point and its fluctuations are nearly twice that of the former.

Similarly, $z'' = (20, 21, 20, 23)$ is also similar to z.

When do we say that two time-series data are similar?

Definition 9.14 Similarity

We say that two time-series X and Y are similar if there exist $a > 0$ and b, such that $y_i = ax_i + b$, for all i.

Distance Metric

Distance metric plays a major role in the measurement of similarity. We say that X and Y are approximately similar with respect to a distance metric d, if $d(X, Y) \leq \varepsilon$ for some tolerance ε. The distance metric can be defined such a way that the scale and shift invariances are taken care of within d. We study this aspect below.

The similarity defined above induces an equivalence relation in the time-series database and hence, the database can be partitioned into equivalent classes. The equivalent class X^* contains all the objects which are similar to X. We can define a unique representation for this class.

Definition 9.15 Normal series

An n-series X is normal if the average $\alpha(X) = 0$, and the deviation $\sigma(X) = 1$.

There exists a unique normal series for X^* and hence the normal form representation of all elements of X^*, is said to be the corresponding normal denoted by $n(X)$. The normal form can play an important role in determining similarity. Two objects are similar with regard to scale and shift, if their corresponding normal forms are similar.

Thus, we can determine similarity between two objects based on the distance between the corresponding normal forms. But computing similarity distance requires $O(n)$ operations. Since the similarity-based query will be required a number of times in a data mining process, it is better to look for an efficient method for computing the similarity value.

Definition 9.16 Discrete Fourier Transform(DFT)

The Discrete Fourier Transform of a n-sequence X is defined to be a sequence of n complex numbers $\mathrm{DFT}_m(X)$, $m = 0, \ldots, (n-1)$, given by

$$\mathrm{DFT}_m(X) = \frac{1}{\sqrt{n}} \sum_{j=1}^{n} x_j e^{-\frac{j(2\pi)m}{n}}$$

where i is the imaginary unit, $\sqrt{(-1)}$. Note that $\mathrm{DFT}_m(X)$ is a complex number.

Definition 9.17 Fingerprint

A fingerprint F(X) of an n-series X is the sequence of first few DFT coefficients of X. In other words,

$$F(X) = (\mathrm{DFT}_1(X), \mathrm{DFT}_2(X), \ldots, \mathrm{DFT}_k(X)),$$

where k assumes a very low value, between 3 and 5.

One interesting property of DFT, as a consequence of Parseval's Theorem, is that

for two similar sequence in terms of the Euclidean distance metric, the corresponding fingerprints are also closer within the specified tolerance.

Thus, the computation of similarity can be restricted only to a few coefficients of DFT.

Longest Common Sequence

A more reasonable similarity notion is based on the longest common subsequence concept, where intuitively X and Y are considered similar if they exhibit similar behaviour for a large part of their length. A similarity definition based on the concept of well-separated geometric sets has been introduced by Bollobas *et al.*, and deterministic and randomized algorithms for computing this similarity measure also have been proposed. Traditionally, the least common subsequence problem can be solved by a dynamic programming problem.

Feature Extraction from Time Series

We can extract k features from every sequence and every sequence is then represented as a unique point in k-dimensional space. Then we can use a multidimensional index to store and search these points. Different multidimensional indexing methods that are currently popular are R^*-tress, k-d Trees, Linear Quad-trees etc. The major problems with these features extraction techniques are:

(a) Completeness of feature extraction: Completeness ensures that similarity between two objects are preserved in the feature space. That is, if two objects are similar in some distance metric, their corresponding maps in k-dimensional feature space should not be grossly dissimilar.
(b) Dimensionality Curse: Most multidimensional indexing scale up for high dimensionality. Such spatial indices do not work well for very high dimensional data.

There are many other distance-preserving orthonormal transformations. These can be categorized into two classes: (a) Data dependent transforms which need all the data to determine the transform, for example, Karhunen-Loeve, SVD, (b) Data-independent transforms, where the transform matrix is determined *a priori*, for example, DFT, DCT, Haar wavelet. DFT does a good job of concentrating energy in the first few coefficients. As we have mentioned earlier, if we keep only the first few coefficients in DFT, we can compute the lower bounds on the actual distance.

The time series segmentation technique partitions a time series into piece-wise linear segments. Each segment has a weight, representing its importance. Weights can be learnt from a training set, in case of supervised learning or else there can be a semiautomatic, user-specified weight adjustment.

Other methods of Analyzing Time Series

The original approach to time series analysis was the establishment of a mathematical model describing the observed system. While linear models were the dominant paradigm for several decades, non-linear models emerged later to deal with systems, showing properties for which linear models are less appropriate. Since the eighties, time-series analysis research has looked towards the exploitation of machine learning algorithms. Those algorithms analyze an unfamiliar time series by learning, that is, by emulating its structure.

Neural networks belong to the domain of supervised learning algorithms and they have been used for the prediction. A known test set can be used to tune the prediction model, so that it can predict the behaviour of an unknown time series in the future.

9.12 Spatial Mining

The immense explosion in geographically referenced data occasioned by developments in IT, digital mapping, remote sensing, and the global diffusion of GIS, emphasizes the importance of developing data mining approaches to geographical analysis and modelling to aid the processes of scientific discovery. Indeed a number of data mining tools are being developed to assist the process of exploring large amounts of data in search of recurrent patterns and relationships.

Spatial data mining is the branch of data mining that deals with spatial (location, or geo-referenced) data. Consider a map of the city of Hyderabad containing various natural and man-made geographic features, and clusters of points (where each point marks the location of a particular house). The houses might be noteworthy because of their size, historical interest, or their current market value. Clustering algorithms exist to assign each point to exactly one cluster, with the number of clusters being defined by the user. We can mine varieties of information by identifying likely relationships. For example, "the land-value of the cluster of residential area to the east of 'Cyber-Tower' is high" or, "70% the Banjara migrants settle in the city around the market area". Such information could be of value to realtors, investors, or prospective home buyers, and also to other domains such as satellite images, photographs, oil and gas explorations. This problem is not trivial—there may be a large number of features to consider. We need to be able to detect relationships among large numbers of geo-referenced objects without incurring significant overheads.

As in the case of temporal data mining, conventional data mining techniques cannot fully exploit the spatial characteristics of data. It is necessary to devise algorithms that take this aspect into consideration.

9.13 Spatial Mining Tasks

The knowledge discovery tasks involving spatial data include finding characteristic rules, discriminant rules, association rules, or deviation and evolution rules, etc. A *spatial characteristic rule* is a general description of spatial data. For example, a rule describing the general price range of houses in various geographic regions in a city is a spatial characteristic rule. A *spatial discriminant rule* is a general description of the features discriminating or contrasting a class of spatial data from other class(es), like the comparison of price ranges of houses in different geographical regions. We shall identify some of the tasks in this section. The following sections deal with other tasks which need detailed study.

Spatial Association Rules

Spatial association rules describe the association between objects, based on spatial neighbourhood relations. We can associate spatial attributes with spatial attributes, or spatial attributes with non-spatial attributes. For example, rules like "The monthly rental of houses around the Cyber-Tower are mostly Rs 500 per sq mt." associates a spatial attribute with a non-spatial attribute. Similarly, a rule like "Uncontrolled tapping of borewell water in the Jubilee Hills area may most likely cause tremors in the neighbouring areas" associates two spatial features.

Attribute-Oriented Induction

Please recall the need of concept hierarchy for attribute-oriented induction. Similarly, the concept hierarchies of spatial and non-spatial attributes can be used to determine relationships between different attributes. This is more relevant in spatial databases. One may be interested in a particular category of land-use patterns. But the classification of land-use types are often hierarchical. A built-up area may be a recreational facility or a residential complex. Similarly, a recreational facility may be a cinema or a restaurant. A water body can be classified into many different categories such as a lake, or a stream. One has to be very specific regarding the level of details to which (s)he wants to discover spatial knowledge.

Aggregate Proximity Relationships

One spatial data mining task is the problem of determining aggregate proximity relationships. This problem is concerned with relationships between spatial clusters based on spatial and non-spatial attributes. Given n input clusters, we want to associate the clusters with classes of features (e.g., educational institutions which, in turn, may

be comprised of secondary schools and junior colleges or higher institutions). The problem is to find classes of features that are in close proximity to most (or all) of the spatial clusters. For example, if 2 of 3 spatial clusters are close to a girls' private schools, and the other cluster is close to a boys' private school, then it can be generalized to the fact that all 3 clusters are close to some private school. Another example would be—"A large banyan tree is normally present within the high-court premises".

Boundary Shape Matching

Another interesting problem can be that of partial boundary shape matching. Specifically, a cluster of points is compared to a large number of natural or man-made features to detect partial or total matches of the facing boundaries of the cluster and features.

The following sections deals with other major mining tasks.

9.14 Spatial Clustering

Different types of spatial clustering algorithms have been proposed. Many of the clustering algorithms discussed in Chapter 5 are directly applicable to spatial data, while some of them may require slight modification in order to be applied to spatial data. For example, GDBSCAN (Recall DBSCAN in Chapter 5) relies on a density-based notion of clusters. The key idea of a density-based cluster is that for each point of a cluster, its epsilon-neighbourhood has to contain at least a minimum number of points. We can generalize this concept in two different ways in the present context. First, any other symmetric and reflexive neighbourhood relationship can be used instead of an epsilon-neighbourhood. While a distance-based neighbourhood is a natural notion of a neighbourhood for point objects, it may be more appropriate to use topological relations such as intersects, meets or above/below to cluster spatially extended objects such as a set of polygons. Second, instead of simply counting the objects in a neighbourhood of an object, other measures to define the "cardinality" of that neighbourhood can be used as well. With these modifications, the DBSCAN algorithm can be used for spatial clustering.

Spatial Characterization

The task of characterization is to find a compact description for a selected subset (the target set) of the database. A spatial characterization is a description of the spatial and non-spatial properties, which are typical for the target objects but not for the whole database. The relative frequencies of the non-spatial attribute values and the relative

frequencies of the different object types are used as the interesting properties. For instance, different object types in a geographic database are mountains, lakes, highways, railroads, etc. To obtain a spatial characterization, not only the properties of the target objects, but also the properties of their neighbours (up to a given maximum number of edges in the relevant neighbourhood graph) are considered. A spatial characterization rule of the form – "Apartments in Sainikpuri have a high rate of retired army officers" – is an example.

9.15 Spatial Trends

A spatial trend is as a regular change of one or more non-spatial attributes, when moving away from a given spatially-referenced object. For instance, "When we move away eastwards from the Cyber-Tower, the rentals of residential houses decrease approximately at the rate of 5% per kilometer". One can also depict the spatial trends pictorially by overlaying the direction of trend on a map. The process is to identify a neighbourhood path starting from a location O and a regression analysis is performed on the respective attribute values for the objects of a neighbourhood path to describe the regularity of change. For the regression, the distance from O is the independent variable and the difference of the attribute values are the dependent variable(s) for the regression. The correlation of the observed attribute values with the values predicted by the regression function, yields a measure of confidence for the discovered trend. The general definition of trend detection for spatial databases is given below.

Definition 9.18 Spatial Trend Detection

Let g be a neighbourhood graph depicting the set of neighbouring points for any given point. Assume that O is an object (node) in g and let a be some subset of non-spatial attributes. We are interested in detecting the changing pattern in a while we move away from O in the neighbourhood graph. Let t be a type of function used for the regression and let *filter* be one of the filters for neighbourhood paths. A filter indicates the subsets of neighbour to be taken into consideration. For example, we may move only in the east to north direction. Let *min-conf* be a real number and let *min-length* as well as *max-length* be natural numbers. The task of spatial trend detection is to discover the set of all neighbourhood paths in g starting from O and having a trend of type t in attributes a, with a correlation of at least *min-conf*. The paths have to satisfy the *filter* and their length must be between *min-length* and *max-length*.

Ester *et al.* propose two algorithms to determine the global trend and local trends.

Algorithm global-trends detects global trends around a start object, O. The existence of a global trend for a start object, O, indicates that if considering all objects on all paths starting from O, the values for the specified attribute(s) in general tend to increase (decrease) with increasing distance.

Spatial Trend Algorithm

```
global-trend(g, O, a, t, min-conf, min-length, max-length, f)
initialize P to be list of all paths in g from O of min-length with filter f;
initialize an empty set of observations;
initialize the last-correlation and last-paths as empty;
initialize first-pos to 1;
initialize last-pos to min-length;
while P ≠ ∅ do
for each p ∈ P do
for every object O_1 from first-pos of p to last-pos of p do
    calculate diff as a(O_1) – a(O) and calculate dist as dist(O_1, O);
    insert the tuple (diff, dist) into the set of observations;
    perform a regression of type t on the set of observations;
    if abs(correlation) of the resulting regression function ≥ min-conf
    then
        set last-correlation to correlation and last-paths to P;
        if the length of the P < max-length
        then
            replace the P by the paths of length 1;
            increment last-pos by 1;
            set first-pos to last-pos;
        else set P to the empty list;
    else return last-correlation and last-paths;
return the last correlation and last-paths;
```

Local-trends technique detects single paths starting from an object O and having a certain trend. The paths starting from O may show different pattern of change; for example, some trends may be positive while the others may be negative.

9.16 Conclusion

In this chapter, we have outlined the current trends in temporal data mining and spatial data mining. We observe that conventional mining techniques are not suitable for temporal and spatial data. Episode discovery, time-series analysis and sequence mining are the most important tasks of temporal mining. Trend analysis, clustering and spatial characterization are the tasks of spatial data mining. There are, in addition, many other areas of temporal and spatial data mining which could not be dealt with in this text.

Further Reading

Roddick's article gives an excellent survey of the techniques on temporal data mining.

Allen [Allen, 83], [Pujari, 1999] and [Vijaya Kumari, 1999] give different qualitative temporal constraints. Temporal association rules are in [Chen, 1998]. Sequence mining was first introduced by Rakesh Agrawal and his team in [Agrawal, 1995] and GSP is proposed in [Srikant, 1996]. However, a general formulation is given by Zaki in [Zaki, 1998] in the context of PLANMINE and SPADE. The other sequence mining algorithm, SPIRIT, was originally proposed in [Garofarakasi, 1999]. WUM was proposed by Weiss in 1998 [Weiss, 1998]. Episode discovery problem was first posed and formulated by Heiki Mannila and his group at the University of Helsinki [Mannila, 1995]. The MEDD and MSDD algorithms [Oates, 1997] discover patterns in multiple event sequences; they explore the rule space directly instead of the sequence space. The similarity of time series is very elegantly presented by Gautam Das and others [Bollobas, 1998]. Sander [Sanders, 1998] gives GDBSCAN for clustering. Ester, in 1998, formulated important spatial data mining problems. Spatial association rules are discussed in [Koperskim 1995].

Exercises

1. Describe the essential features of temporal data and temporal inferences.
2. Formulate the sequence mining problem.
3. Discuss the major algorithms of the sequence mining problem.
4. Diseuss the essential features of the GSP algorithm.
5. What is the event-prediction problem? Propose one algorithm to solve this problem.
6. What are different tasks of time-series mining?
7. Describe different similarity measures of time-series data.
8. Discuss the major features of the timeweaver algorithm.
9. Describe the working of the SPADE algorithm.
10. Relate the two problems "Association rules with item constraint" and the "problem class handled by the SPIRIT algorithm".
11. How do you distinguish spatial mining from temporal mining?
12. Is it possible to use BIRCH for spatial clustering? If so, devise a method for spatial clustering using BIRCH.
13. How do you handle spatial and non-spatial data, while carrying out any mining task?
14. Propose different neighbourhood relationships that can be used for density-based clustering of spatial data.
15. Why is concept hierarchy important for spatial data? Identify certain cases of occurrences of such hierarchies in spatial data.
16. How do you handle spatial or temporal data for decision tree construction?

BIBLIOGRAPHY

Aho A.V. Algorithms for finding patterns in strings. In *Handbook of Theoretical Computer Science*, Volume A: Algorithms and Complexity, Elsevier, pp. 255–400, 1990.

Al-Naemi S. A theoretical framework for temporal knowledge discover. In *Proceedings of the International Workshop on Spatio-temporal Databases*, pp. 23–33, 1994.

Allen J.F. Maintaining knowledge about temporal intervals, *Commun. ACM*, **26**:11, 832–843, 1983.

Brockwell P.J., and Davis R. *Introduction to Time Series and Forecasting*. Springer-Verlag, 1996.

Brendt D.J., and Clifford J. Finding patterns in time-series: A Dynamic programming approach. In *Advances in Knowledge Discovery and Data Mining*, AAAI Press, 1995.

Chen M.-S., Park J.S., and Yu P.S. Efficient data mining for path traversal patterns. *IEEE Transactions on Knowledge and Data Engineering*, **10**(2), March 1998, pp. 209–221.

Chen X., and Petrounias I. A framework for temporal data mining. In *Proceedings of DEXA'98, LNCS-1460*, Springer-Verlag, 1998.

Das G., Gunopulos D., and Mannila H. Finding similar time series. Manuscript, 1996.

Davison B., and Hirsh H. Probabilistic online action prediction. In *Proceedings of the AAAI Spring Symposium on Intelligent Environments*, 1998.

Dietterich T., and Michalski R. Discovering patterns in sequences of events. *Artificial Intelligence*, **25**, 1985, pp. 187–232.

Ester M., Frommelt A., Kriegel H-P, and Sander J. Algorithm for characterization and trend detection in spatial databases. *Fourth KDD Conference*, 1998.

Ester M., Kriegel H.-P., and Sander J. Knowledge discovery in spatial databases, In *Proceedings of 23rd German Conference on AI(KI'99)*, Bonn, Germany, 1999.

Faloutos C., Ranganathan M., and Manolopoulos Y. Fast subsequence matching in time-series databases. In *SIGMOD'94*, 1994.

Garofalakis M.N., Rastogi R., and Shim K. SPIRIT: Sequential pattern mining with regular expression constraints. *Bell Labs Tech. Memorandum* BL0112370-990223-03TM, February 1999.

Giordana A., Saitta L., and Zini F. Learning disjunctive concepts by means of genetic algorithms. In *Proceedings of the 11th International Conference on Machine Learning*, 1994, pp. 96–104.

Goldin D.Q., and Kanellakis P.C. On similarity queries for timeseries data: constraint specification and implementation.

Jagadish H.V., Mendelzon A.O., and Milo T. Similarity-based queries. In *Proceedings of the 14th Symposium on Principles of Database Systems (PODS'95)*, pp. 36–45, 1995.

Koperski K., and Han J. Discovering spatial association rules in geographic information databases. In *Proceedings of the 4th International Synposis on Large Spatial Databases*, pp. 47–66, 1995.

Mannila H., Toivonen H., and Verkamo A.I. Discovering frequent episodes in sequences. In *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining*, August 1995.

Ng R.T. Spatial data mining: Discovering knowledge of clusters of maps. In *Proceedings SIGMOD Workshop*, 1996.

Ng R.T., Lakshmanan L.V.S., Han J., and Pang A. Exploratory mining and pruning optimizations of constrained association rules. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, June 1998.

Openshaw S. Two exploratory space-time attribute pattern analyzers relevant to GIS. S Fotheringham and P Rogerson (eds.) *GIS and Spatial Analysis*, Taylor and Francis, London, pp. 83–104, 1994.

Openshaw S. Developing automated and smart spatial pattern exploration tools for geographical information systems applications. The Statistician **44**, pp. 3–16, 1995.

Openshaw S. Building automated geographical analysis and exploration machines in P.A. Longley, S.M. Brooks, R. Mcdonnell, B. Macmillan (eds.) *Geocomputation: A Primer*, Wiley Chichester, pp. 95–115, 1998.

Pujari A.K., and Sattar A. A new framework of reasoning about points, interval and durations. In *Proceedings of IJCAI'99*, Sweden, pp. 1259–1265, 1999.

Roddick J.F., and Spiliopoulou M. *Temporal Data Mining: Survey and Issues.*

Sander J., Ester M., Kriegel H.-P., and Xu X. Density based clustering in spatial databases. A new algorithm and its applications, *Data Mining and Knowledge Discovery, an International Journal*, **2**, No 2, 1998.

Sasisekharan R., Seshadri V., and Weiss S. Data mining and forecasting in large-scale telecommunication networks. *IEEE Expert*, **11**(1),1996, pp. 37–43.

Shatkay H., and Zdonik S. Approximate queries and representations for large data sequences. In *ICDE'96*, 1996.

Spiliopoulou M., and Faulstich L.C. WUM-A tool for web utilization analysis. In *EDBT Workshop WebDB'98, LLNCS-1590*, Springer-Verlag, 1999.

Srikant R., and Agrawal R. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology (EDBT'96)*, March 1996.

Srikant R., Vu Q., and Agrawal R. Mining association rules with item constraints. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, August 1997.

Vijaya K., Pujari A.K., and Sattar A., INDU-Interval and duration network, In *the Proceedings of Australian AI Conference*, Sydney, Australia, 1999.

Wang J.T.-L., Chirn G.-W., Marr T.G., Shapiro B., Shasha D., and Zhang K. Combinatorial pattern discovery for scientific data: Some Preliminary Results. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, May 1994.

Weiss G.M., and Hirsh H. Learning to predict rare events in event sequences. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, AAAI Press, 1998, pp. 359–363.

White D.A., and Jain R. Algorithms and strategies for similarity retrieval. *Technical Report VCL-96-101*. Visual Computing Laboratory, UC Davis, 1996.

Yazdani N., and Ozsoyoglu Z.M. Sequence matching of images. In *Proceedings of the 8th International Conference on Scientific and Statistical Database Management*, 1996, pp. 53–62.

Zaki N.J., Lesh N., and Ogihara M. PLANMINE: sequence mining for plan failures. In *KDD'98*, 1998.

INDEX

S

T

U

V

W

X

Related Titles from Universities Press (India) Ltd.

Agarwal, A.K. & Dhananjay Pershad	*Mastering PC Software: Lotus 1-2-3, WordStar and dBASE III PLUS*
Arthur Anderson & EIU	*Managing Business Risks in the Information Age*
Billings, C.W.	*Supercomputers: Shaping the Future*
Datta, N.	*Computer Programming and Numerical Analysis*
Dillon, P.M. & Leonard, D.C.	*Multimedia and the Web from A to Z*
Ganesan, V.	*Computer Simulation of Compression-Ignition Engine Processes*
Ghoshal, S.K.	*A Practical Approach to Parallel Computing*
Grady, S.M.	*Virtual Reality: Computers Mimic the Physical World*
Henderson, H.	*Communication and Broadcasting*
Henninger, M.	*Don't Just Surf: Effective Research Strategies for the Net*
Jagadeesh, T.R. et al.	*Computer Laboratory Referral for Engineering and Diploma Students*
Kaujalgi, V.B.	*Structured Systems Analysis and Design: Data Flow Approach*
Kippenhahn, R.	*Code Breaking: A History and Exploration*
Lockwood, T. & Scott, K.	*A Writer's Guide to the Internet*
Lynch, P.J. & Horton, S.	*Web Style Guide: Basic Design Principles for Creating Web Sites*
Mohapatra, P.K.J., *et al.*	*Introduction to System Dynamics Modeling*
Naughton, J.	*A Brief History of the Future: The Origins of the Internet*
Prabhu, C.S.R.	*Semantic Database Systems: A Functional Introduction*
Rajaraman, V.	*Supercomputers*
Shiva Ramu, S.	*Cyberspace and the Repositioning of Corporations*
Smith, Steve	*Thinking About Computer Programming?*
Thro, E.	*Robotics: The Marriage of Computers and Machines*
Verma, Surendra	*Computer, Internet and Multimedia Dictionary*

Data Mining Techniques addresses all the major and latest techniques of data mining and data warehousing. It deals in detail with the latest algorithms for discovering association rules, decision trees, clustering, neural networks and genetic algorithms. The book contains the algorithmic details of different techniques such as *A priori*, Pincer-search, Dynamic Itemset Counting, FP-Tree growth, SLIQ, SPRINT, BOAT, CART, RainForest, BIRCH, CURE, BUBBLE, ROCK, STIRR, PAM, CLARANS, DBSCAN, GSP, SPADE, SPIRIT, etc. Interesting and recent developments such as Support Vector Machines and Rough Set Theory are also covered in the book. The book also discusses the mining of web data, spatial data, temporal data and text data. An elaborate bibliography and exercises are provided at the end of each chapter. This book can serve as a textbook for students of computer science, mathematical science and management science. It can also be an excellent handbook for researchers in the area of data mining and data warehousing.

Arun K Pujari is a Professor of Computer Science at the University of Hyderabad, Hyderabad. Prior to joining UoH, he served at the Automated Cartography Cell, Survey of India, and Jawaharlal Nehru University, New Delhi. He received his PhD from the Indian Institute of Technology, Kanpur and MSc from Sambalpur University, Sambalpur. He has also undertaken several visiting assignments at the Institute of Industrial Sciences, University of Tokyo; International Institute of Software Technology, United Nations University, Macau; University of Memphis, USA; and Griffith University, Australia, among others.

Rs 350.00

ISBN 81 7371 380 4
9 788173 713804

Universities Press